U.S. Government Protection Profile for Separation Kernels in Environments Requiring High Robustness

VERSION 0.621

Information Assurance Directorate

National Security Agency
9800 Savage Road
Fort George G. Meade, MD 20755-6000

1 July 2004

This page intentionally left blank.

Foreword

1 This publication, "*U.S. Government Protection Profile for Separation Kernels in Environments Requiring High Robustness*", is issued by the Information Assurance Directorate as part of its program to promulgate security standards for information systems. This protection profile is based on the "Common Criteria for Information Technology Security Evaluations, Version 2.1." [1]

2 Comments on this document should be directed to: ppcomments@iatf.net. The comments should include the title of the document, the page, the section number, and paragraph number, detailed comment and recommendations.

Table of Contents

List of Figures

List of Tables

1. Introduction

3 This section contains overview information necessary to allow a Protection Profile (PP) to be registered through a Protection Profile Registry. The PP identification provides the labeling and descriptive information necessary to identify, catalogue, register, and cross-reference a PP. The PP overview summarizes the profile in narrative form and provides sufficient information for a potential user to determine whether the PP is of interest. The overview can also be used as a stand-alone abstract for PP catalogues and registers. The "Conventions" section provides the notation, formatting, and conventions used in this protection profile. The "Glossary of Terms" section gives a basic definition of terms, which are specific to this PP. The "Document Organization" section briefly explains how this document is organized.

1.1 Identification

4 Title: U.S. Government Protection Profile for Separation Kernels in Environments Requiring High Robustness

5 Registration: Information Assurance Directorate

6 Keywords: separation kernel, COTS, high robustness, data isolation, information flow control, partition, cryptography

1.2 Overview

7 This "*U.S. Government Protection Profile for Separation Kernels in Environment Requiring High Robustness*" (SK PP) specifies the security functional and assurance requirements for a class of Separation Kernels. Unlike the traditional Security Kernel that performs all trusted functions for a secure operating system, a Separation Kernel's primary security function is to partition (viz. separate) the subjects and resources of a system into policy-based equivalence classes, and to control information flows between partitions. The partitions and information flow policies are defined by the Separation Kernel's configuration data. A Separation Kernel evaluated against this PP provides the trusted foundation for use in security critical and complex applications whose security requirements are not addressed by this PP.

8 This SK PP uses Department of Defense (DoD) and National Information Assurance (IA) guidance and policies as a basis to establish the requirements for National Security Systems[1]. Products meeting this protection profile become candidates for use in National Security Systems. However, compliance to this protection profile is not, by itself, sufficient.

9 Conformant products support *information flow control, secure initialization, trusted delivery,*

[1] National Security Systems are systems that contain classified information or involves intelligence activities, involves cryptologic activities related to national security, involves command and control of military forces, involves equipment that is an integral part of a weapon or weapon system, or involves equipment that is critical to the direct fulfillment of military or intelligence missions.

trusted recovery and *audit* capabilities. Compliance alone does not offer sufficient confidence that national security information is appropriately protected in the context of a larger system in which the TOE is integrated. Designers of such systems must apply appropriate systems security engineering principles and techniques to afford acceptable protection for national security information. In particular, it is the responsibility of the larger system's designers to articulate support for a coherent application-level security policy in the Separation Kernel's configuration data, as well as to ensure that the configuration data itself is coherent and self-consistent. It is only with well-formed configuration data that the Separation Kernel can be expected to enforce mission critical policies. Such policies may include those that are associated with the management of information classified at different sensitivity levels based on its degree of confidentiality or integrity.

10 Conformant products may also be suitable for commercial mission-critical applications, given similarly well-formed configuration data. The judgement as to whether a given instantiation of configuration data is coherent, as well as being well formed with respect to some application-level security policy are beyond the scope of this protection profile. It is noted that this protection profile applies specific environmental assurance requirements on certain external programs (e.g., applications, libraries or middleware) in systems where the configuration data set defines partitions that are not partially ordered with respect to information flow control.

11 This protection profile also levies assurance requirements to the support tools and procedures that are critical to developing and operating the Separation Kernel but which are not considered as part of the trusted security functions of the Separation Kernel.

1.2.1 TOE Environment Defining Factors

12 The environment for a TOE can be characterized by the authorization (or lack of authorization) of the least trustworthy entity compared to the highest value of TOE resources (i.e. the TOE itself and all of the data processed by the TOE).

13 In trying to specify the environments in which TOEs with various levels of robustness are appropriate, it is useful to first discuss the two defining factors that characterize the environment: value of the resources and authorization of the entities to those resources.

14 Note that there are an infinite number of combinations of entity authorization and value of resources; this conceptually "makes sense" because there are an infinite number of potential environments, depending on how the resources are valued by the organization, and the variety of authorizations the organization defines for the associated entities. In the next section 1.2.2, these two environmental factors will be related to the robustness required for selection of an appropriate TOE.

1.2.1.1 Value of Resources

15 Value of the resources associated with the TOE includes the data being processed or used by the TOE, as well as the TOE itself (for example, a real-time control processor). "Value" is assigned by the organization that owns the resources. For example, in the DoD low-value data might be

equivalent to data marked "FOUO", while high-value data may be those classified Top Secret. In a commercial enterprise, low-value data might be the internal organizational structure as captured in the corporate on-line phone book, while high-value data might be corporate research results for the next generation product.

16 Note that when considering the value of the data one must also consider the value of data or resources that are accessible through exploitation of the TOE. For example, a firewall may have "low value" data itself, but it might protect an enclave with high value data. If the firewall was being depended upon to protect the high value data, then it must be treated as a high-value-data TOE.

1.2.1.2 Authorization of Entities

17 Authorization that entities (users, administrators, other IT systems) have with respect to the TOE (and thus the resources of that TOE, including the TOE itself) is an abstract concept reflecting a combination of the trustworthiness of an entity and the access and privileges granted to that entity with respect to the resources of the TOE. For instance, entities that have total authorization to all data on the TOE are at one end of this spectrum; these entities may have privileges that allow them to read, write, and modify anything on the TOE, including all TSF data. Entities at the other end of the spectrum are those that are authorized to few or no TOE resources. For example, in the case of an operating system, an entity may not be allowed to log on to the TOE at all (that is, they are not valid users listed in the operating system's user database).

18 It is important to note that authorization **does not** refer to the **access** that the entities actually have to the TOE or its data. For example, suppose the owner of the system determines that no one other than employees is authorized to certain data on a TOE, yet they connect the TOE to the Internet. There are millions of entities that are **not authorized** to the data (because they are not employees), but they actually have connectivity to the TOE through the Internet and thus can attempt to access the TOE and its associated resources.

19 Entities are characterized according to the value of resources to which they are authorized; the extent of their authorization is implicitly a measure of how trustworthy the entity is with respect to any of the applicable security policies.

1.2.2 Selection of Appropriate Robustness Levels

20 Robustness is a characteristic of a TOE defining how well it can protect itself and its resources; a more robust TOE is better able to protect itself. This section relates the defining factors of IT environments, authorization, and value of resources to the selection of appropriate robustness levels.

21 When assessing any environment with respect to Information Assurance, the critical point to consider is the likelihood of an attempted security policy compromise, which was characterized in the previous section in terms of entity authorization and resource value. As previously mentioned, robustness is a characteristic of a TOE that reflects the extent to which a TOE can

protect itself and its resources. It follows that as the likelihood of an attempted resource compromise increases, the robustness of an appropriate TOE should also increase.

22 It is critical to note that several combinations of the environmental factors will result in environments in which the likelihood of an attempted security policy compromise is similar. Consider the following two cases:

23 The first case is a TOE that processes only low-value data. Although the organization has stated that only its employees are authorized to log on to the system and access the data, the system is connected to the Internet to allow authorized employees to access the system from home. In this case, the least trusted entities would be unauthorized entities (e.g. non-employees) exposed to the TOE because of the Internet connectivity. However, since only low-value data are being processed, the likelihood that unauthorized entities would find it worth their while to attempt to compromise the data on the system is low and selection of a basic robustness TOE would be appropriate.

24 The second case is a TOE that processes high-value (e.g., classified) information. The organization requires that the TOE be in a closed environment, and that every user with physical and logical access to the TOE undergo an investigation so that they are authorized to the highest value data on the TOE. Because of the extensive checks done during this investigation, the organization is assured that only highly trusted users are authorized to use the TOE. In this case, even though high value information is being processed, it is unlikely that a compromise of that data will be attempted because of the authorization and trustworthiness of the users; therefore, selection of a basic robustness TOE would be appropriate.

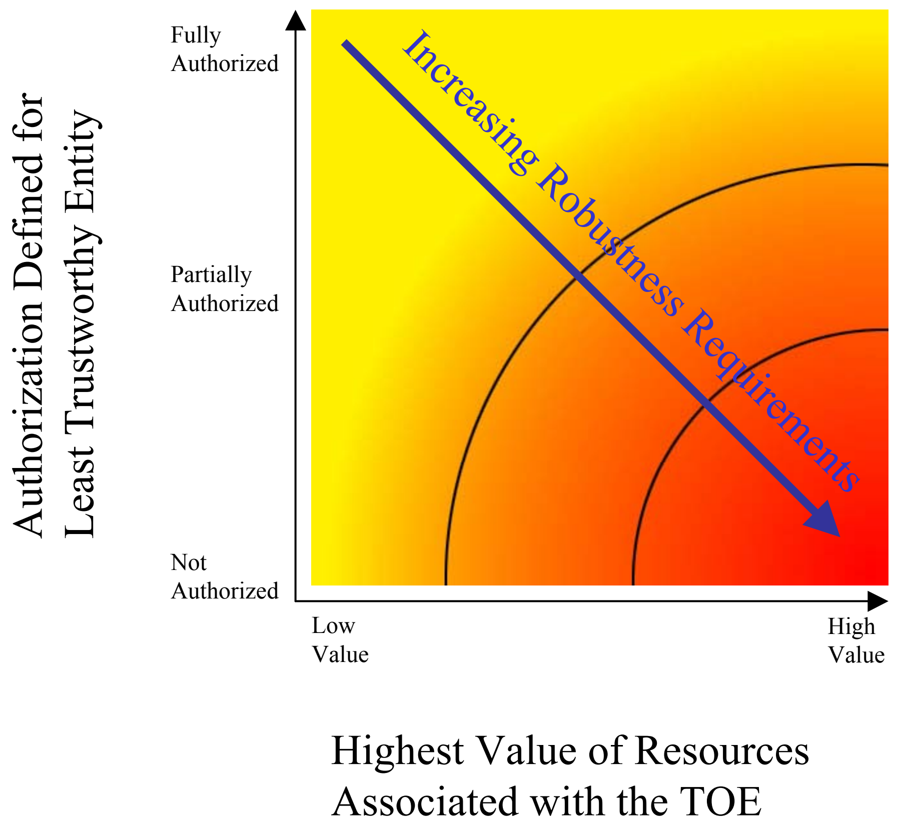

Authorization Defined for Least Trustworthy Entity
Fully Authorized
Partially Authorized
Not Authorized
Increasing Robustness Requirements
Low Value
High Value
Highest Value of Resources Associated with the TOE

Figure 1-1 Universe of Environments

25 The preceding examples demonstrated that it is possible for radically different combinations of entity authorization and resource values to result in a similar likelihood of an attempted compromise. As mentioned earlier, the robustness of a system is an indication of the protection being provided to counter compromise attempts. Therefore, a basic robustness system should be sufficient to counter compromise attempts where the likelihood of an attempted compromise is low. Figure 1-1 depicts the "universe" of environments characterized by the two factors discussed in the previous section: on one axis is the authorization defined for the least trustworthy entity, and on the other axis is the highest value of resources associated with the TOE.

26 As depicted in Figure 1-1, the robustness of the TOEs required in each environment steadily increases as one goes from the upper left of the chart to the lower right; this corresponds to the need to counter increasingly likely attack attempts by the least trustworthy entities in the environment. Note that the shading of the chart is intended to reflects the notion that different environments engender similar levels of "likelihood of attempted compromise", signified by a similar color. Further, the delineations between such environments are not stark, but rather are finely grained and gradual.

27 While it would be possible to create many different "levels of robustness" at small intervals along the "Increasing Robustness Requirements" line to counter the increasing likelihood of

attempted compromise due to those attacks, it would not be practical nor particularly useful. Instead, in order to implement the robustness strategy where there are only three robustness levels: Basic, Medium, and High, the graph is divided into three sections, with each section corresponding to a set of environments where the likelihood of attempted compromise is roughly similar. This is graphically depicted in the Figure 1-1.

28 A second representation of environments is shown in Figure 1-2, the "dots" represent given instantiations of environments; like-colored dots define environments with a similar likelihood of attempted compromise. Correspondingly, a TOE with a given robustness should provide sufficient protection for environments characterized by like-colored dots. In choosing the appropriateness of a given robustness level TOE PP for an environment, then, the user must first consider the lowest authorization for an entity as well as the highest value of the resources in that environment. This should result in a "point" in the chart above, corresponding to the likelihood that that entity will attempt to compromise the most valuable resource in the environment. The appropriate robustness level for the specified TOE to counter this likelihood can then be chosen.

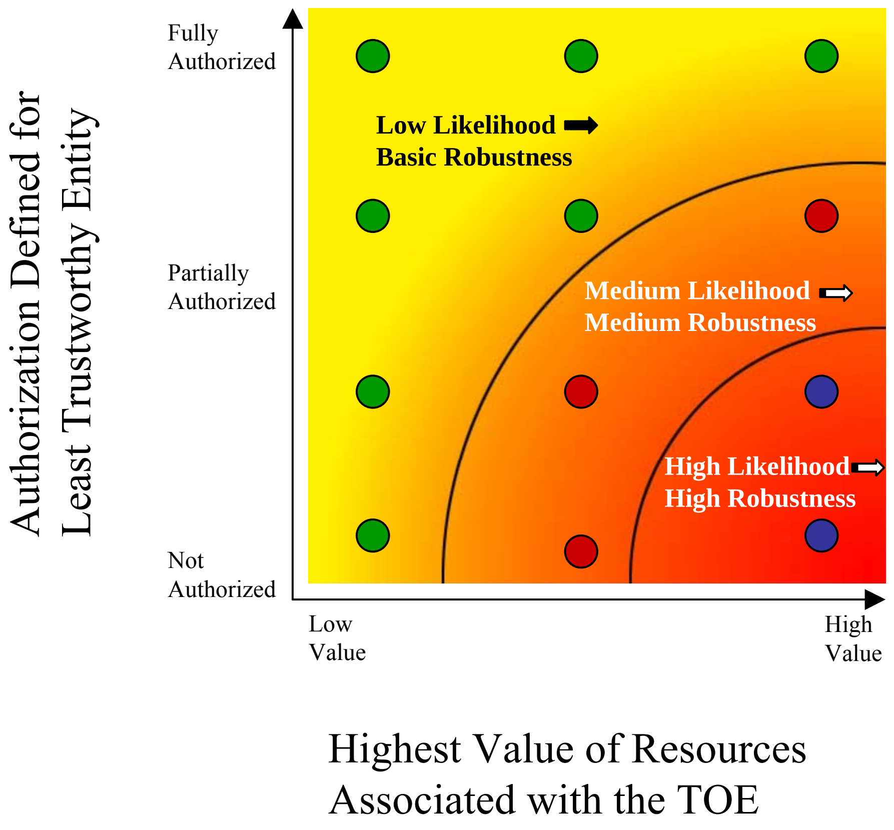
29 
Authorization Defined for Least Trustworthy Entity
Fully Authorized
Partially Authorized
Not Authorized
Low Likelihood ➡ Basic Robustness
Medium Likelihood ➡ Medium Robustness
High Likelihood ➡ High Robustness
Low Value
High Value
Highest Value of Resources Associated with the TOE

Figure 1-2 Likelihood of Attempted Compromise

30 The difficult part of this activity is differentiating the authorization of various entities, as well as determining the relative values of resources; (e.g., what constitutes "low value" data vs. "high value" data). Because every organization will be different, a rigorous definition is not possible. In section 3 of this PP, the targeted threat level for a high robustness TOE is characterized. This

information is provided to help organizations using this PP ensure that the functional requirements specified by this high robustness PP are appropriate for their intended application of a compliant TOE.

1.3 Mutual Recognition of Common Criteria Certificates

31 The assurance requirements contained in this PP are equivalent to the Evaluated Assurance Level 6 (EAL 6) as defined in the Common Criteria (CC) [3] with augmentation. The augmented assurances are in the areas of development, independent testing, systematic flaw remediation, and maintenance of assurance. COTS Separation Kernels meeting the requirements of this profile provide a high level of robustness. Under the “Arrangement on the Mutual Recognition of Common Criteria Certificates in the field of Information Technology Security” document, only CC requirements at or below EAL 4 are mutually recognized. Because this profile exceeds the limits imposed by the “Arrangement on the Mutual Recognition of Common Criteria Certificates in the field of Information Technology Security” document, the US will recognize only certificates issued by the US evaluation scheme to meet this profile. Other national schemes are likewise under no obligation to recognize US certificates with assurance components exceeding EAL4.

1.4 Conventions

32 The notation, formatting, and conventions used in this protection profile (PP) are consistent with version 2.1 of the Common Criteria for Information Technology Security Evaluation. Font style and clarifying information conventions were developed to aid the reader.

33 The CC permits four functional component operations: assignment, iteration, refinement, and selection to be performed on functional requirements. These operations are defined in Common Criteria, Part 2, paragraph 2.1.4 as:

- assignment: allows the specification of an identified parameter;
- refinement: allows the addition of details or the narrowing of requirements;
- selection: allows the specification of one or more elements from a list; and
- iteration: allows a component to be used more than once with varying operations.

34 *Assignments or selections* left to be specified by the developer in subsequent security target documentation are italicized and identified between brackets (“[ ]”). In addition, when an assignment or selection has been left to the discretion of the developer, the text “assignment:” or “selection:” is indicated within the brackets. Assignments or selection created by the PP author (for the developer to complete) are bold, italicized, and between brackets (“[ ]”). CC selections completed by the PP author are underlined and CC assignments completed by the PP author are bold.

35 *Refinements* are identified with “**Refinement:**” right after the short name. They permit the addition of extra detail when the component is used. The underlying notion of a refinement is

that of narrowing. There are two types of narrowing possible: narrowing of implementation and narrowing of scope[2]. Additions to the CC text are specified in bold. Deletions of the CC text are identified in the "End Notes" with a bold number after the element ("**8**").

36 *Iterations* are identified with a number inside parentheses ("(#)"). These follow the short family name and allow components to be used more than once with varying operations.

37 *Explicit Requirements* are allowed to create requirements should the Common Criteria not offer suitable requirements to meet the PP needs. The naming convention for explicit requirements is the same as that used in the CC. To ensure these requirements are explicitly identified, the ending "_EXP" is appended to the newly created short name. However, most of the explicit requirements are based on existing CC requirements. To make it easier for the PP reader to view any changes from existing CC elements, the added text is written in bold text.

38 *Application Notes* are used to provide the reader with additional requirement understanding or to clarify the author's intent. These are italicized and usually appear following the element needing clarification.

39 Table 1.1 provides examples of the conventions (explained in the above paragraphs) for the permitted operations.

Table 1.1 – Functional Requirements Operation Conventions

Convention	Purpose	Operation
Bold	The purpose of bolded text is used to alert the reader that additional text has been added to the CC. This could be an assignment that was completed by the PP author or a refinement to the CC statement. Examples: FDP_IFC.2.1 The TSF shall enforce **Information Flow Control policy** on **subjects and all resources** and on all operations that cause information to flow to and from subjects covered by the SFP. FAU_GEN.1.1 **Refinement:** The TSF shall be able to generate **audit data for** the following auditable events:	(Completed) Assignment or Refinement

[2] US interpretation #0362: Scope of Permitted Refinements

Convention	Purpose	Operation
Italics	The purpose of italicized text is to inform the reader of an assignment or selection operation to be completed by the developer or ST author. It has been left as it appears in the CC requirement statement. Examples: FAU_ARP.1.1 The TSF shall take [assignment: *list of least disruptive actions*] upon detection of a potential security violation. FDP_RIP.2.1 The TSF shall ensure that any previous information content of a resource is made unavailable upon the *[selection: allocation of the resource to, deallocation of the resource from]* all exported resources.	Assignment (to be completed by developer or ST author) or Selection (to be completed by developer or ST author)
<u>Underline</u>	The purpose of underlined text is to inform the reader that a choice was made from a list provided by the CC selection operation statement. Example: FAU_SEL.1.1 The TSF shall be able to include or exclude auditable events from the set of audited events based on the following attributes: **a)** <u>subject identity,</u> **b)** <u>event type,</u> **c) success of auditable security events, and** **d) failure of auditable security events.**	Selection (completed by PP author)

Convention	Purpose	Operation
Parentheses (Iteration #)	The purpose of using parentheses and an iteration number is to inform the reader that the author has selected a new field of assignments or selections with the same requirement and that the requirement will be used multiple times. Iterations are performed at the component level. The component behavior name includes information specific to the iteration between parentheses. Example: **5.4.1.1 Explicit: Management of TSF Data (for Configuration Data) (FMT_MTD_EXP.1)** FMT_MTD_EXP.1.1(1) The TSF shall restrict the ability to select and activate the TSF policy configuration data to authorized subjects. **5.4.1.1 Explicit: Management of TSF Data (for General TSF Data) (FMT_MTD_EXP.1(2))** FMT_MTD_EXP.1.1(2) The TSF shall prevent modification of TSF policy configuration data.	Iteration 1 (of component) Iteration 2 (of component)
Explicit: (**_EXP**)	The purpose of using **Explicit:** before the family or component behavior name is to alert the reader and to explicitly identify a newly created component. To ensure these requirements are explicitly identified, the "**_EXP**" is appended to the newly created short name and the component and element names are bolded. Example: **5.4.1.1 Explicit: Management of Security Functions Behavior (FMT_MOF_EXP.1)** **FMT_MOF_EXP.1.1** The TSF shall restrict the ability to enable and disable audit generation to the configuration data.	Explicit Requirement

Convention	Purpose	Operation
Endnotes	The purpose of endnotes is to alert the reader that the author has deleted Common Criteria text. An endnote number is inserted at the end of the requirement, and the endnote is recorded on the last page of the section. The endnote statement first states that a deletion was performed and then provides the rationale. Following is the family behavior or requirement in its original and modified form. A strikethrough is used to identify deleted text and bold for added text. A text deletion rationale is provided. Examples: Text as shown: FAU_SAA.1.2 **Refinement:** The TSF shall monitor the accumulation or combination of the following events known to indicate a potential security violation: [assignment: *subset of defined auditable events*].1 Endnote statement: 1 A deletion of CC text was performed in FAU_SAA.1.2. Rationale: The words "enforce the following rules for monitoring audited events: a)" were deleted for clarity and flow of the requirement. Additionally the assignment was moved from the middle of the requirement to the end for clarity and flow of the requirement. FAU_SAA.1.2 **Refinement:** The TSF shall ~~enforce the following rules for monitoring audited events: a)~~ **monitor the** accumulation or combination of ~~[assignment: *subset of defined auditable events*]~~ **the following events** known to indicate a potential security violation: [assignment*: subset of defined auditable events*].	Refinement

1.5 Glossary of Terms

40 This profile includes terms from the Common Criteria [1] by reference. Other terms are described in this section to aid in the understanding and application of the requirements.

Administrator	Any personnel responsible for the configuration, installation, maintenance, or integration of the TOE or its data.
Configuration Function	The procedures and automated mechanisms employed to generate the TSF configuration data and corresponding integrity seals.

Configuration data Configuration data, flow policy Configuration data, non-flow policy	*TSF data that* provides various control information to the TSF that is used by the TSF during initialization to define the secure initial state and its behavior during runtime. Configuration data consists of flow policy configuration data and supporting policy configuration data. See Appendix D. TSF *flow policy configuration data* assigns (*binds*) subjects and exported resources to partitions, thereby defining TSF partitions, and also defines the information flow control and partition flow control policies for determining communication between and within those partitions. TSF *non-flow policy configuration data* defines all other configurable TSF critical data, such as audit configuration parameters, cryptographic configuration parameters, clock settings and execution period for self-test.
Covert Channel	An unintended and/or unauthorized communications path that can be used to transfer information in a manner that violates a security policy [6]. Covert channels allow transfer of information through indirect access by subjects to internal resources; whereas, a transfer of information in violation of the security policy through exported resource(s) would be a *TSF flaw*.
Delivery, Trusted	Procedures and automated mechanisms employed to deliver a copy of the TOE from the TOE developer to the customer, such that the customer copy is assured to be the same as the developer's master copy.
Initialization Function	The procedures and automated mechanisms that bring the TSF to an *initial secure state*. Initialization includes the boot function that brings the TSF implementation (code) and TSF data (e.g., configuration data) into its execution domain (e.g., read it from disk, from ROM, or from flash memory into a memory space allocated for TSF functions and data).
Load Function	The procedures and automated mechanisms to convert the software implementation and/or configuration data into a TSF-useable form. The initial load function can take different forms, including: placement of the implementation or configuration information onto suitable media (e.g., CD, ROM or flash memory); or compilation of configuration data as part of the TSF implementation. The load function may also include the insertion or installation of the media into the TOE hardware at either the TOE developer or customer site.

Maintenance Mode	A system mode initiated by the detection of a TSF failure or imminent TSF failure in which some or all of the security functions are non-operational, and either: 1) the TSF can recover automatically; or 2) user intervention is required to recover.
Operational Mode	A mode of operation where all of the TSF security functions are available and all SFPs are enforced.
Partition	A set of subjects and a set of exported resources that are within the same policy-based equivalence class as defined by the configuration data. For a given partition, either but not both sets may be empty. Note that a partition is not an active entity: see *subject*. Resources that are by default accessible by all partitions are virtualized and exported. The configuration data assigns (binds) each *exported resource* to a single partition for the purposes of defining such partitions. Every subject is assigned (*bound*) to a single partition by the configuration data for the purposes of defining partitions.
Principle of Least Privilege	This principle requires that each subject and internal module in a system be granted the most restrictive set of privileges (or lowest clearance) needed for the performance of authorized tasks. The application of this principle limits the damage that can result from accident, error, or unauthorized use [5].
Residual Information Protection (RIP)	Ensuring that deleted information is no longer accessible, and that newly created exported resources do not contain information that should not be accessible [2].
Resource Resource, Exported Resource, Internal	*Resources* are the totality of all hardware, firmware and software and data that are executed, utilized, created, protected or exported by the TSF (separation kernel). *Exported resources* are those resources to which an explicit reference is possible via a TSF interface (e.g., the programming or configuration interface). *Internal resources* are those resources that are not exported resources.
Secure State Secure State, Initial	A state defined in the TSP model in which the TOE has consistent TSF data and the TSF can correctly enforce the policy. [2, par. 1236] The *initial secure state* is the secure state arrived at after a successful initialization.

Separation Kernel	Hardware and/or firmware and/or software mechanisms whose primary function is to separate multiple partitions and control information flow between and within the partitions.
Subject	An active entity within the TSC that causes operations to be performed. A subject is an abstraction created by the TSF and exported at the TSFI.
TSF Data	Data created by and for the TOE that might affect the operation of the TOE. See Appendix D for more information.
Trusted Individual	A person or persons who perform procedures upon which the security of the TOE may depend. Requirements for identification, authentication, and establishing the trustworthiness of trusted individuals are allocated to the IT environment. See OE.TRUSTED_INDIVIDUAL. **Note:** The current document assumes that trusted individuals interact with the TOE before runtime; considering the above definition, is it reasonable to also allow this interaction during runtime and/or maintenance mode?

1.6 Document Organization

41 *Section 1* provides the introductory material for the protection profile.

42 *Section 2* describes the Target of Evaluation in terms of its envisaged usage and connectivity.

43 *Section 3* defines the expected TOE security environment in terms of the threats to its security, the security assumptions made about its use, and the security policies that must be followed.

44 *Section 4* identifies the security objectives derived from the threats and policies.

45 *Section 5* identifies and defines the security functional requirements from the CC that must be met by the TOE in order for the functionality-based objectives to be met.

46 *Section 6* identifies the TOE security assurance requirements.

47 *Section 7* provides a rationale to explicitly demonstrate that the information technology security objectives satisfy the policies and threats. Arguments are provided for the coverage of each policy and threat. The section then explains how the set of requirements are complete relative to the objectives, and that each security objective is addressed by one or more component requirements. Arguments are provided for the coverage of each objective.

48 *Section 8* identifies background material used as reference to create this profile.

49 *Appendix A* defines frequently used acronyms.

50 *Appendix B* identifies cryptographic standards, policies and other publication referenced in this PP.

51 *Appendix C* provides a rationale for the IFC/IFF requirements.

52 *Appendix D* provides a description of the various types of TSF data.

53 *Appendix E* provides a rationale for changes to ADV requirements.

54 *Appendix F* provides an example scenario for TOE functions.

2. Target of Evaluation (TOE) Description

2.1 Product Type

55 This protection profile specifies requirements for a multi-partition separation kernel together with a uni-processor hardware base for use as a component in National Security Systems and other systems responsible for protecting highly sensitive information. The separation kernel divides all resources under its control into partitions such that the actions of an active entity (i.e., a subject) in one partition are isolated from (viz., cannot be detected by or communicated to) an active entity in another partition, unless an explicit means for that communication has been established. To achieve high robustness and other rigorous verification objectives, the separation kernel must be minimized with respect to functionality, architecture and design. This protection profile focuses on core functional capabilities that include:

- Partitioning (separation and isolation) of all resources, including CPU, memory and devices
- Controlled sharing of selected resources
- Audit services

56 The separation kernel provides to its hosted software programs high-assurance partitioning and information flow control properties that are both tamperproof and non-bypassable. These capabilities provide a trusted foundation upon which the enforcement of specific application-level (vs. kernel-level) security policies can be achieved. Examples of these software programs include multilevel secure reference monitors, guards, device drivers, file managers, and message-passing services, as well as those for implementing operating system, middleware and virtual machine monitor abstractions.

2.2 Separation Kernel Concepts

57 A separation kernel achieves isolation of subjects in different partitions by virtualization of shared resources: each partition encompasses a resource set that appears to be entirely its own (see Figure 2-1). To achieve this objective for resources that can only be accessed by one subject at a time, such as the CPU, the ideal separation kernel must ensure that the temporal usage patterns from different partitions are not apparent to each other. Other resources, such as memory, may be accessed by different partitions simultaneously, while preserving idealized isolation, if the separation kernel ensures, for example, that partitions are allocated different and non-interacting portions of the resource. Furthermore, separation kernel utilization of its own internal resources must also preserve the desired isolation properties.

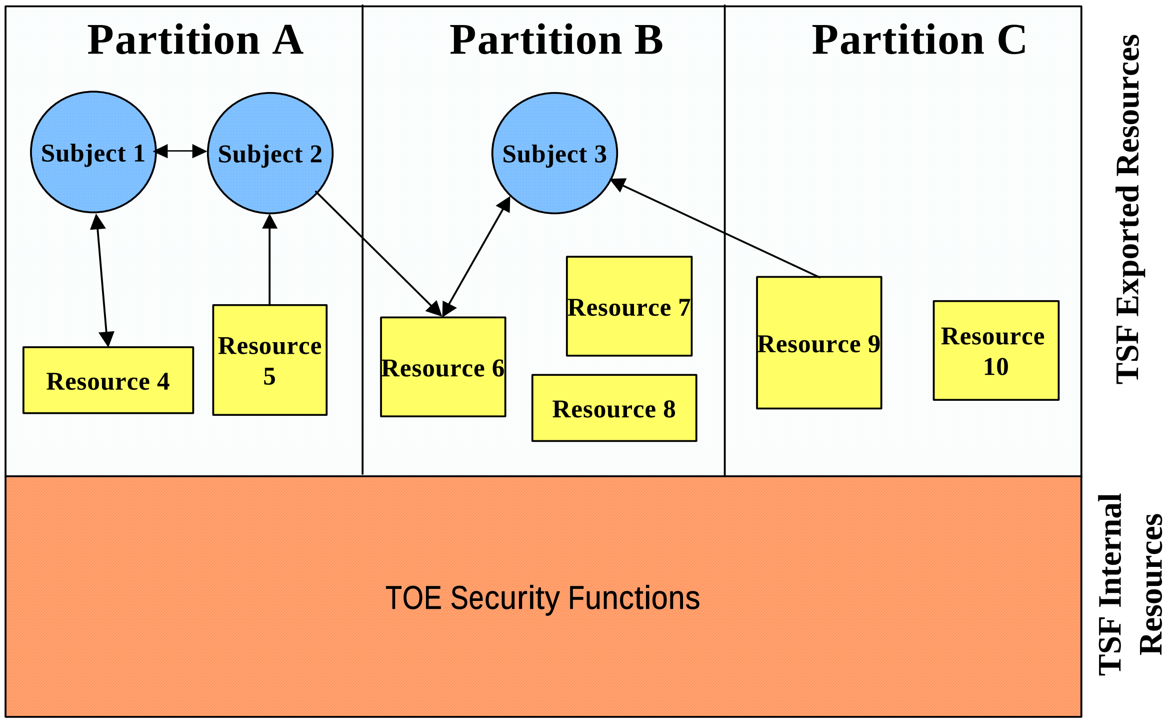


Figure 2-1. SKPP Resource Abstraction

58 A separation kernel meeting these properties will have *configuration data* that establishes the static partition definitions and the static allocation of resources to partitions. The configuration data is created using evaluated tools and procedures provided as part of the TOE. After it is created, the configuration data may be *loaded* onto various media (e.g., PROM or CD) for later *initialization* into the TSF, or it may be encoded directly into the TSF, e.g., as part of the delivered software implementation or hardware.

59 Figure 2-2 describes the allocation of separation kernel components to the TSF and TOE. These components are involved in several steps leading to an initial secure state, as follows (see also example TOE scenario in Appendix F):

- Trusted Delivery. The TOE developer employs trusted mechanisms and procedures to deliver the TOE, or implementation components of the TOE to the customer (e.g., system integrator, application developer or end user). Trusted delivery is used for the initial version as well as updates. See also Figure 2-5.

- Configuration. A trusted individual employs evaluated tools and procedures to generate the machine-readable configuration data and a corresponding integrity seal. At a minimum, the configuration tool must translate human-readable (e.g., ASCII) characters into machine-readable (e.g., binary) format. A trusted individual verifies the accuracy of the generated configuration data either through manual inspection or with the support of trusted automated tools.

- Load. A trusted individual employs evaluated tools and procedures to transfer the software implementation and configuration data, either together or separately, into a form that is accessible by TOE. Loading of configuration data may occur in the customer IT environment and/or in the TOE developer IT environment. A developer "load" step (if any) occurs before trusted delivery. The load function may also be included as part of offline Trusted Recovery (see "halted" state in Figure 2-6) in the event that the configuration data needs to be recovered.

- Initialization. A trusted individual or IT mechanism initiates the functions for starting the TSF, e.g. via a power-on switch or other mechanism accessible to the IT environment. Then, a trusted individual may need to employ further evaluated procedures (e.g., in a non-embedded environment), after or during which the initialization functions complete the transformation of the TSF into a secure initial state. The initialization functions include verification of the integrity of the TSF code and configuration data, as well as the transfer of that code and data into the TSF execution domain.

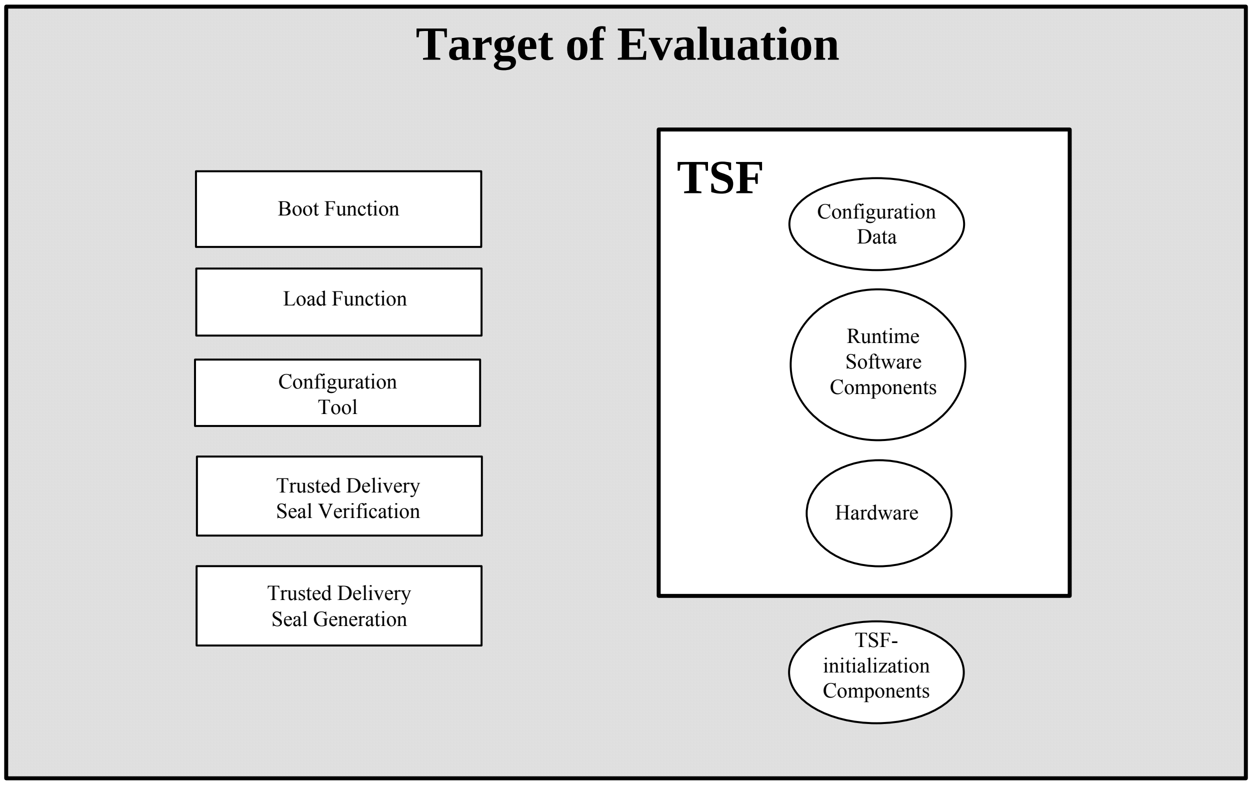

Target of Evaluation
Boot Function
Load Function
Configuration Tool
Trusted Delivery Seal Verification
Trusted Delivery Seal Generation
TSF
Configuration Data
Runtime Software Components
Hardware
TSF-initialization Components

Figure 2-2 Allocation of Separation Kernel Components

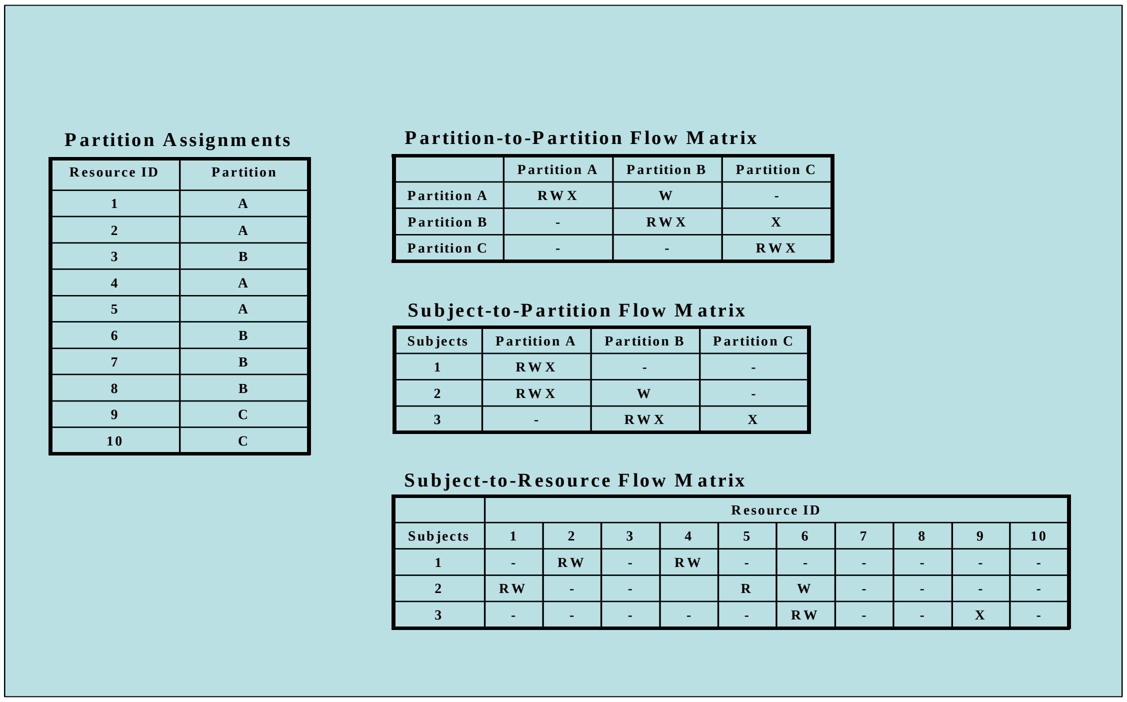
60 A partition is an abstraction implemented by the TSF from resources under its control. The configuration data will include the assignment of active subject entities (e.g., programs, tasks, processes, threads) and externally available resources to partitions, as well as information as to how (e.g., in which mode) partitions and/or resources may be accessed by subjects and/or other

partitions. A given partition may have zero, one or more subjects, as well as zero or more resources assigned to it.

61 The content and format of the configuration data may take different forms, depending on the scope and granularity of the information flow control and partition flow control policies to be enforced, as well as on other factors of system design. Figure 2-3 shows example configuration data structures required to represent the partitions and flows illustrated in Figure 2-1.[3] A given system may be designed to enforce the information flow policy at the granularity of the partition (see "Partition-to-Partition Flow Matrix," in Figure 2-3), and/or at the granularity of subjects and resources within partitions (see "Subject-to-Resource Flow Matrix," in Figure 2-3), and/or some combination of these (see "Subject-to-Partition Flow Matrix," in Figure 2-3).

62 The Subject-to-Resource flow matrix of Figure 2-3 represents an application of the principle of least privilege with respect to the partitions and flows shown in Figure 2-1: no subject is given more authorization than the minimal required to invoke operations to achieve the required flows. An alternate approach to achieving the principle of least privilege for a system that enforces only a Partition-to-Partition policy is through the use of a minimal configuration. A minimal configuration of a system is where, for all pairs of partitions, X and Y, all of the subjects of partition X require for their functionality, and are given, the same authorization to invoke operations on all of the resources of partition Y. Note that X and Y may be the same or different partitions, and the access given may be empty for a given pair of partitions. A minimal configuration is required for this alternate approach because the Partition-to-Partition matrix may otherwise provide more authorization than what is required. Figure 2-4 shows a system (e.g., for transaction processing) with a minimal configuration.

[3] Notice that *all* intra-partition flows are also explicitly allowed in the two "partition" flow matrices to reflect an intra-partition free-flow policy, but such "self" flows may also be inhibited, or expressed implicitly, e.g., by convention.



Partition Assignments

Resource ID	Partition
1	A
2	A
3	B
4	A
5	A
6	B
7	B
8	B
9	C
10	C

Partition-to-Partition Flow Matrix

	Partition A	Partition B	Partition C
Partition A	RWX	W	-
Partition B	-	RWX	X
Partition C	-	-	RWX

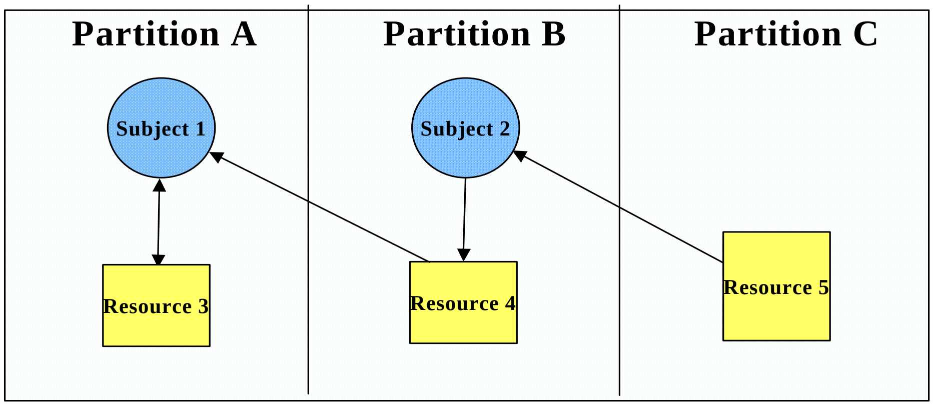
Subject-to-Partition Flow Matrix

Subjects	Partition A	Partition B	Partition C
1	RWX	-	-
2	RWX	W	-
3	-	RWX	X

Subject-to-Resource Flow Matrix

	Resource ID									
Subjects	1	2	3	4	5	6	7	8	9	10
1	-	RW	-	RW	-	-	-	-	-	-
2	RW	-	-		R	W	-	-	-	-
3	-	-	-	-	-	RW	-	-	X	-

Figure 2-3 Example SKPP Configuration Data



Partition-to-Partition Flow Matrix

	Partition A	Partition B	Partition C
Partition A	RW	R	-
Partition B	-	W	R
Partition C	-	-	-

Figure 2-4 Minimal Configuration of Resources

63 While the information flow control and partition flow control policies, and resource allocations

are largely fixed during runtime, the protection profile allows a separation kernel to have a flexible mechanism for emergency modification of policy and resource allocation. This mechanism allows several complete sets of policy configuration data to be initialized into the separation kernel, one of which will define the *current policy*. Authorized subjects may have the ability to select and activate a different current policy during runtime.

2.3 Trusted Delivery

64 The TOE and components of the TOE may be distributed in various ways, both for the initial delivery and for subsequent updates. This protection profile requires that the TOE include procedures and/or tools to verify that the on-site version of the TOE matches the master version (see "Trusted Delivery" in Figure 2-5). Such a verification tool may be configured to execute on the TOE (but not as part of the TSF) or on other hardware, but in either case the tool and the hardware that it runs on are evaluated as part of the TOE. Verification of the TOE and configuration data occur again as part of initialization. Figure 2-5 shows how applications and TOE configuration data may be installed by the TOE developer and/or by various entities within the customer environment. However, the TOE as well as any separately delivered TOE components must be delivered to the customer environment by means of trusted delivery. The end result is that the "user" will have a complete and verified TOE, with verified configuration data and installed applications. If TOE components were modified after trusted delivery, then the TOE would not be in an evaluated configuration.

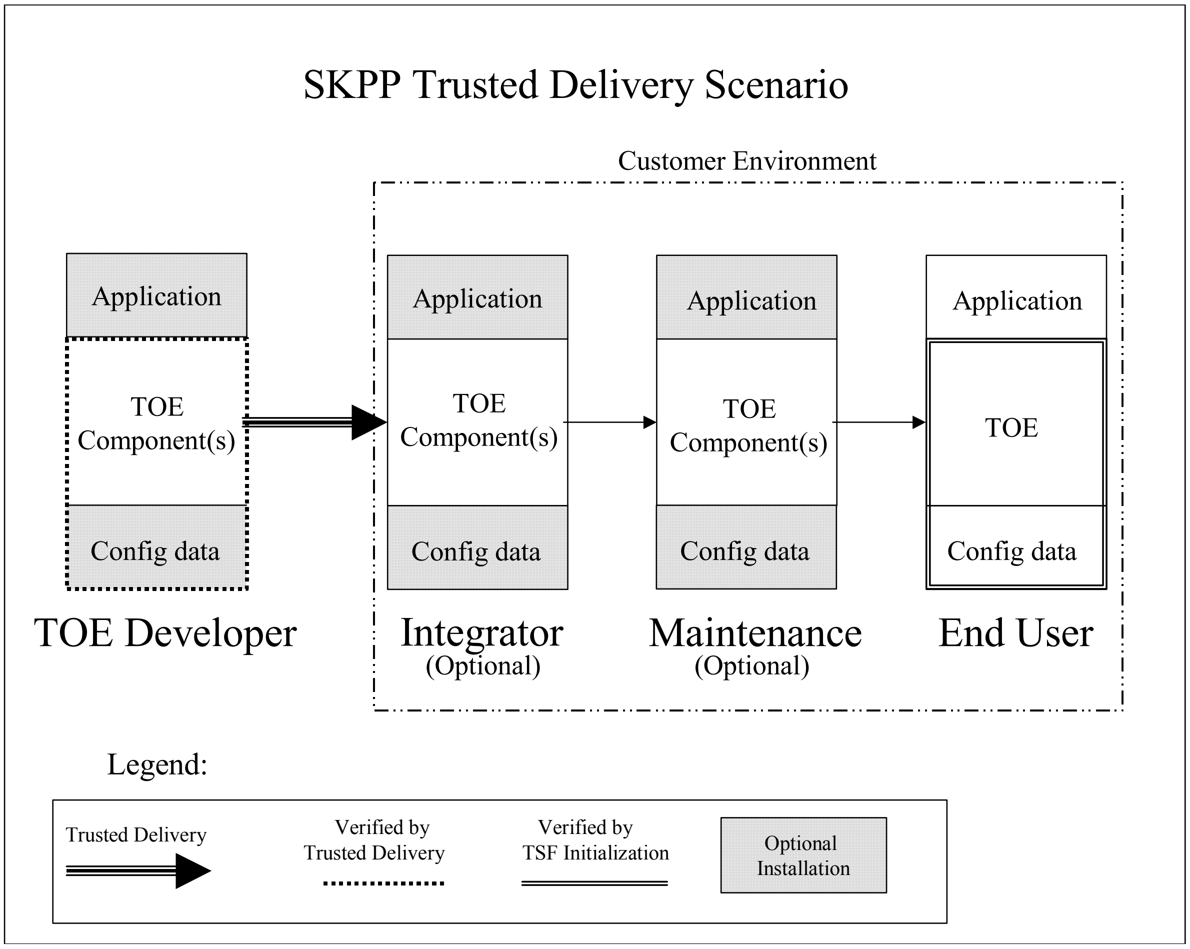


Figure 2-5 Trusted Delivery Scenario

2.4 Trusted Recovery

65 At a given time, the separation kernel will be in one of three modes of operation: Halted, Operational or Maintenance. Similarly, the separation kernel could be in either a secure state or an insecure state. Table 2-1 summarizes the possible mode/state combinations:

	Secure	Insecure
Operational	√	√
Maintenance	√	n/a
Halted	√	n/a

Table 2-1. Possible mode/state combinations

66 Figure 2-6 illustrates a generic recovery scenario involving these modes and states. A successful

initialization brings the kernel to an operational secure state. For failures that require preservation of secure state, as specified by FPT_FLS.1, the kernel must remain in the operational secure state while handling the failures. Recovery may occur directly within the operational secure state, or indirectly via the (secure) maintenance mode. The implementation of the fail-secure mechanism is ST-specific.

67 For failures that do not require preservation of secure state under FPT_FLS.1, the separation kernel may temporarily enter an operational insecure state. That is, between the time that a security failure first occurs and the time that the TSF can detect it and respond, the conservative assumption is that the failure introduces insecurity. This insecure state is ephemeral because the separation kernel will return to the operational secure state if the failure is either directly recoverable, or indirectly recoverable via the (secure) maintenance mode.

68 In maintenance mode, some or all of the TSF interface functions (TSFIs) are unavailable. This restriction to a degraded or a completely non-operational, runtime functionality may enable the TSF to establish and maintain a secure state during the remaining recovery sequence. This protection profile does not require interaction with a trusted individual to return from maintenance mode to secure operational mode, as some other protection profiles might (e.g., see [2] Volume 2, paragraph 1234). The implementation of maintenance mode is ST-specific.

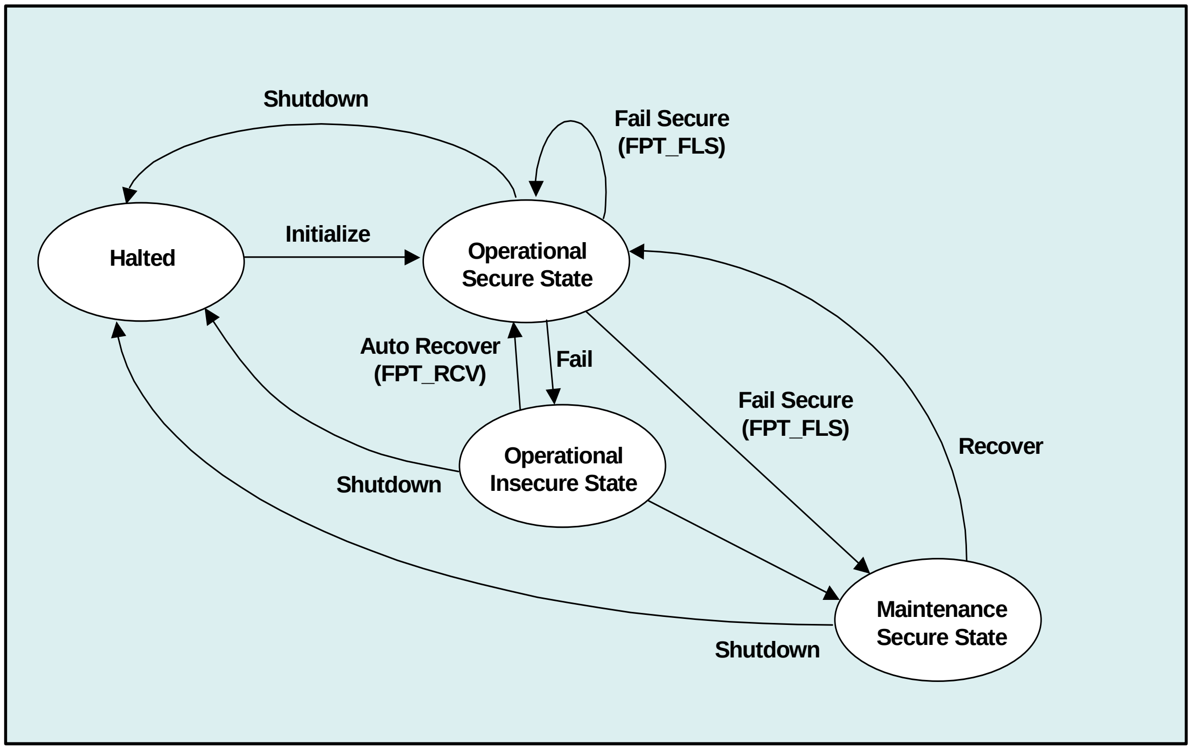


Figure 2-6 TOE Recovery State Diagram

2.5 Evaluation Considerations

69 A separation kernel may be structured or configured to function within a larger system as an embedded or a non-embedded component. From an evaluation standpoint, the critical difference between embedded and non-embedded functionality is the degree to which trust is distributed between authorized individuals and the component during runtime. In the non-embedded separation kernel, the security provided during runtime may depend on the procedural actions of authorized individuals to monitor, maintain, or otherwise manage the separation kernel; in the embedded case, there can be no distribution of trust to authorized individuals during runtime, so these security functions are performed solely by the separation kernel. Thus, this protection profile may include certain runtime requirements that may be satisfied by either automated security functions or manual procedures.

70 Additionally, a separation kernel, like many systems under Common Criteria evaluation, may include hardware or software components that have been created by different developers. While this heterogeneous diversity is not a conceptual problem for evaluation, it may present constraints on the execution of the evaluation process, such as the order of the evaluation of components. Even if diverse TOE developers create the various components for a given separation kernel, each TOE component must be evaluated to the requirements of this protection profile as part of that TOE.

71 Similarly, the modular and component structure of separation kernels may differ. The TOE (i.e., separation kernel) may consist of separate initialization and runtime components; separate hardware independent and hardware dependent components[4]; as well as the cross product of these two dichotomies (see Table 2-2). This structure will be a critical factor in the separation kernel evaluation, especially with regard to whether a module or component is determined to be in the TSF, the TOE, or in the IT environment. Initialization components are not part of the TSF and are evaluated under separate criteria (see ADV_INI), whereas all runtime components are part of the TSF. The evaluation requirements for the TSF, TOE and IT environment are separately stated elsewhere in this document.

	Hardware Dependent	Hardware Independent
Initialization	√	√
Runtime	√	√

Table 2-2: Types of Components in a Separation Kernel

[4] Examples of hardware dependent components are an "architecture support package" for interaction with a specific processor and a "board support package" (BSP) for interaction with a specific processor environment (devices, buses, I/O, etc.).

72 This protection profile applies to both embedded and non-embedded separation kernels, as well as homogeneously- and heterogeneously-developed separation kernels. However, it is outside of the scope of this protection profile to address how the evaluation process or methodology is affected by these differences. This protection profile takes a standard approach to address different or unique requirements for different separation kernel configurations, as follows:

- When possible, differences in the criteria are first addressed through use of the CC-defined operations of assignment, selection, iteration and refinement;
- In the cases where the CC-defined operations do not suffice, the CC-defined *explicitly stated requirements* model is used;
- Where there are differences in the implications of the criteria, those differences are addressed by the Application Notes that follow the criteria.

73 When this TOE is used in composition with other components or products to make up a larger system environment, it is the responsibility of the larger system's designers to articulate support for a coherent application-level security policy in the separation kernel's configuration data, as well as to ensure that the configuration data itself is coherent and self-consistent. It is only with well-formed configuration data that the separation kernel can be expected to enforce mission critical policies. The judgment as to whether a given instantiation of configuration data is self-consistent, or well-formed with respect to the intended application-level security policy, is beyond the scope of this protection profile and beyond the scope of the evaluation of the separation kernel.

74 Encryption functions utilized for the security and integrity of this TOE and the information it protects must be implemented with appropriate cryptographic strengths and assurances as approved by NSA to ensure adequate protection.

2.6 General TOE Functionality

75 Conformant separation kernels must include the following security features:

- Information flow control to enforce strict partition isolation with the exception of explicit interactions allowed via TSF (configuration) data
- Cryptographic services which provide mechanisms to protect the integrity of TSF code and data as it resides within the TOE and when it is transmitted to other trusted external components; and
- Initialization into a secure state and trusted recovery from a failure to a secure state
- Failure detection and response; and
- Generation of audit data in support of middleware and application level audit services.

76 Among the requirements not levied in this PP are:

- User interfaces for operational mode, maintenance mode or initialization;
- Identification and Authentication which mandates authorized users to be uniquely identified and authenticated by the TSF;
- Discretionary Access Control (DAC) which restricts access to objects based on the identity of subjects and/or groups to which they belong, and allows authorized users to specify protection for objects that they control;
- Cryptographic services for applications to encrypt, decrypt, hash, and digitally sign data as it resides within the system and as it is transmitted to other systems;
- Complete physical protection mechanisms, which must be provided by the environment.

2.7 Cryptographic Requirements

77 The TOE cryptographic services must provide both a level of functionality and assurance regardless of its implementation (software, hardware, or any combination thereof). This is achieved by meeting both the NIST FIPS PUB 140-2 standard and all additional requirements as stated in this PP (refer to Appendix B for relevant cryptographic standards, policies, and other publications).

78 For cryptographic services fully implemented in hardware, all FIPS PUB 140-2 Level 3 requirements excluding Roles, Services and Authentication, as well as all additional requirements identified in this PP, must be met. For all other implementations (i.e., software, or a combination of software and hardware), all the requirements identified in FIPS PUB 140-2 Security Level 1 excluding Roles, Services and Authentication, plus some of the requirements for FIPS PUB 140-2 Security Level 3 (namely, those in the areas of: Cryptographic Module Ports and Interfaces; Cryptographic Key Management; and Design Assurance); and all additional requirements identified in this PP must be met. These two implementations, with the exception of the Electromagnetic Interference/Electromagnetic Compatibility requirements, are equivalent in intent and counter the identified threats in this protection profile.

79 For convenience, Section 5.2 of this PP identifies where a NIST certification is required and against what standard. To meet this PP, the developer must have a NIST certification and receive NSA approval for compliance to Section 5.2 and all other crypto-related requirements in this PP.

2.8 TOE Operational Environment

80 It is assumed that the TOE environment is under the control of a single administrative authority and has a homogeneous system security policy, including personnel and physical security. This environment can be specific to an organization or a mission and may also contain multiple networks or enclaves. Enclaves may be logical or be based on physical location and proximity.

81 The TOE may be accessible by external IT systems that are beyond the environment's security policies. The users of these external IT systems are similarly beyond the control of the separation kernel's policies. Although the users of these external systems are authorized in their

environments, they are outside the scope of control of this particular environment so nothing can be presumed about their intent. They must be viewed as potentially hostile.

3. TOE Security Environment

82 This section defines the expected TOE security environment in terms of the threats, security assumptions, and the security policies that must be followed for the high robustness TOE.

3.1 Use of High Robustness

83 A high robustness TOE is considered necessary protection for environments where the likelihood of an attempted compromise is high, and the value of the protected resources is high. This implies that the motivation of the threat agents will be high. Note that this also implies that the resources and expertise of the threat agents may be high, because highly sophisticated threat agents may be motivated to use great expertise and extensive resources in an environment where high robustness is suitable.

84 An alternative perspective to thinking of the robustness level in terms of "likelihood of attempted compromise" is to consider the damage to the organization that would result if a TOE compromise were to occur. These two notions (likelihood of compromise and damage resulting from compromise) are parallel notions. They both are intrinsically linked to the value of the data being processed. The more valuable/sensitive the data, the greater the likelihood that an adversary will attempt to compromise the TOE, similarly the greater the damage to the organization that would result from such compromise.

3.2 Threat Agent Characterization

85 In addition to helping define the robustness appropriate for a given environment, the threat agent is a key component of the formal threat statements in the PP. Threat agents are typically characterized by a number of factors such as *expertise, available resources, and motivation.* Because each robustness level is associated with a variety of environments, there are corresponding varieties of specific threat agents (that is, the threat agents will have different combinations of motivation, expertise, and available resources) that are valid for a given level of robustness. The following discussion explores the impact of each of the threat agent factors on the ability of the TOE to protect itself (that is, the robustness required of the TOE).

86 The *motivation* of the threat agent seems to be the primary factor of the three characteristics of threat agents outlined above. Given the same expertise and set of resources, an attacker with low motivation may not be as likely to attempt to compromise the TOE. For example, an entity with no authorization to low value data nonetheless has low motivation to compromise the data because of its low value; thus a basic robustness TOE should offer sufficient protection. Likewise, fully authorized users with access to highly valued data similarly have low motivation to attempt to compromise the data because of their authorization, thus again a basic robustness TOE should be sufficient.

87 Unlike the motivation factor, however, the same can't be said for *expertise*. A threat agent with low motivation and low expertise is just as unlikely to attempt to compromise a TOE as an attacker with low motivation and high expertise; this is because the attacker with high expertise does not have the motivation to compromise the TOE even though they may have the expertise

to do so. The same argument can be made for *resources* as well.

88 Therefore, when assessing the robustness needed for a TOE, the motivation of threat agents should be considered a "high water mark". *That is, the robustness of the TOE should increase as the motivation of the threat agents increases.*

89 Having said that, the relationship between expertise and resources is somewhat more complicated. In general, if resources include factors other than just raw processing power (money, for example), then expertise should be considered to be at the same "level" (low, medium, high, for example) as the resources because money can be used to purchase expertise. Expertise in some ways is different, because expertise in and of itself does not automatically procure resources. However, it may be plausible that someone with high expertise can procure the requisite amount of resources by virtue of that expertise.

90 It may not make sense to distinguish between these two factors; in general, it appears that the only effect these may have is to lower the robustness requirements. For instance, suppose an organization determines that, because of the value of the resources processed by the TOE and the trustworthiness of the entities that can access the TOE, the motivation of those entities would be "medium". This normally indicates that a medium robustness TOE would be required because the likelihood that those entities would attempt to compromise the TOE to get at those resources is in the "medium" range. However, now suppose the organization determines that the entities (threat agents) that are the least trustworthy have no resources and are unsophisticated. In this case, even though those threat agents have medium motivation, the likelihood that they would be able to mount a successful attack on the TOE would be low, and so a basic robustness TOE may be sufficient to counter that threat.

91 It should be clear from this discussion that there is no "cookbook" or mathematical answer to the question of how to specify exactly the level of motivation, the amount of resources, and the degree of expertise for a threat agent so that the robustness level of TOEs facing those threat agents can be rigorously determined. However, an organization can look at combinations of these factors and obtain a good understanding of the likelihood of a successful attack being attempted against the TOE. Each organization wishing to procure a TOE must look at the threat factors applicable to their environment; discuss the issues raised in the previous paragraph; consult with appropriate accreditation authorities for input; and document their decision regarding likely threat agents in their environment.

92 The important general points to make are:

- The motivation for the threat agent defines the upper bound with respect to the level of robustness required for the TOE
- A threat agent's expertise and/or resources that is "lower" than the threat agent's motivation (e.g., a threat agent with high motivation but little expertise and few resources) may lessen the robustness requirements for the TOE (see next point, however).

93 The availability of attacks associated with high expertise and/or high availability of resources (for example, via the Internet or "hacker chat rooms") introduces a problem when trying to define the expertise of, or resources available to, a threat agent.

3.3 Threats

94 The following threats are addressed by PP compliant TOEs:

T.ADMIN_ERROR	An administrator may incorrectly install or configure the TOE (including the misapplication of the principle of least privilege to limit the damage that can result from accident, error, or unauthorized use), or install a corrupted TOE resulting in ineffective security mechanisms.
T.ALTERED_DELIVERY	The TOE may be corrupted or otherwise modified during delivery such that the on-site version does not match the master distribution version.
T.BAD_RECOVERY	The TOE may be placed in an insecure state as a result of unsuccessful recovery from a system failure or discontinuity.
T.COVERT_CHANNEL_EXPLOIT	An unauthorized information flow may occur between partitions as a result of covert channel exploitation.
T.CRYPTO_COMPROMISE	A malicious subject may cause key, data or executable code associated with the cryptographic functionality to be inappropriately accessed (viewed, modified, or deleted), thus compromising the cryptographic mechanisms and the data protected by those mechanisms.
T.INCORRECT_BOOT	The TSF implementation and TSF data are not correctly transferred into the TSF's execution domain.
T.INCORRECT_CONFIG	The TSF configuration data does not accurately reflect the user's intentions regarding partitioning and information flow.
T.INCORRECT_LOAD	The TSF code and/or configuration data are not correctly converted into a TSF-useable form.
T.INSECURE_STATE	When the TOE is initially started or restarted after a failure, the security state of the TOE may be in an insecure state.
T.POOR_DESIGN	Unintentional or intentional errors in requirements specification or design of the TOE may occur, leading to flaws that may be exploited by a malicious subject.

T.POOR_IMPLEMENTATION	Unintentional or intentional errors in implementation of the TOE design may occur, leading to flaws that may be exploited by a malicious subject.
T.POOR_TEST	Lack of or insufficient tests to demonstrate that all TOE security functions operate correctly (including in a fielded TOE) may result in incorrect TOE behavior being undiscovered.
T.RESIDUAL_DATA	A subject may gain unauthorized access to data through reallocation of TOE resources from one subject to another.
T.RESOURCE_EXHAUSTION	A malicious subject may block others from system resources (e.g., system memory, persistent storage, and processing time) via a resource exhaustion attack.
T.TSF_COMPROMISE	A malicious subject may cause TSF data or executable code to be inappropriately accessed (viewed, modified or deleted).
T.UNAUTHORIZED_ACCESS	A subject may gain access to resources or services for which it is not authorized according to the TOE security policy.

3.4 Security Policy

95 The following organizational security policies are addressed by PP compliant TOEs:

P.ACCOUNTABILITY	The TOE shall provide the capability to make available information regarding the occurrence of security relevant events.
P.CRYPTOGRAPHY	The TOE shall use NIST FIPS validated cryptography as a baseline with additional NSA-approved methods for key management (i.e., generation, access, distribution, destruction, handling, and storage of keys) and for cryptographic operations (i.e., encryption, decryption, signature, hashing, key exchange, and random number generation services).
P.INDEPENDENT_TESTING	The TOE must undergo independent testing.
P.LEAST_PRIVILEGE	The TOE shall be designed such that the principle of least privilege is applied to limit the damage that can result from accident, error or unauthorized use.

P.RATINGS_MAINTENANCE	A plan for procedures and processes to maintain the TOE's rating must be in place to maintain the TOE's rating once it is evaluated.
P.SELECT_POLICY	The TOE shall provide the capability to select and activate a complete set of new policy configuration data. The TOE shall ensure that the policy enforced upon the activation of the new configuration data is consistent with the new configuration data
P.SYSTEM_INTEGRITY	The TOE shall provide the ability to periodically validate its correct operation and, with the help of administrators if necessary, it must be able to recover from any errors that are detected.
P.USER_GUIDANCE	The TOE shall provide documentation regarding the correct use of the TOE security features.
P.VULNERABILITY_ANALYSIS_AND_TEST	The TOE must undergo independent vulnerability analysis and penetration testing by NSA to demonstrate that the TOE is resistant to an attacker possessing a high attack potential.

3.5 Security Usage Assumptions

96 The specific conditions below are assumed to exist in a PP-compliant TOE environment:

A.CHANNELS	If the residual risk from covert channels is a concern, it is assumed that the applications executing on the TOE are trusted with assurance commensurate with the value of the IT assets protected by the TOE.
A.PHYSICAL	It is assumed that the IT environment provides the TOE with appropriate physical security commensurate with the value of the IT assets protected by the TOE.
A.TRUSTED_FLOWS	If a subject is allowed by the configuration data to cause information flow in violation of the partial ordering of information flows between partitions, it is assumed that the subject is trusted with assurance commensurate with the value of the IT assets in all partitions to which it has access.
A.TRUSTED_INDIVIDUAL	If an individual is allowed to perform procedures upon which the security of the TOE may depend, it is assumed that the individual is trusted with assurance commensurate with the value of the IT assets.

4. Security Objectives

97 This section defines the security objectives for the TOE and its environment. These objectives are suitable to counter all identified threats and cover all identified organizational security policies and assumptions. The TOE security objectives are identified with “O.” appended to the beginning of the name and the environment objectives are identified with “OE.” Appended to the beginning of the name.

4.1 TOE Security Objectives

O.ACCESS	The TOE will ensure that subjects gain only authorized access to resources that it controls.
O.ADMIN_GUIDANCE	The TOE will provide administrators with the necessary information for secure management of the TOE.
O.AUDIT_GENERATION	The TOE will provide the capability to detect and generate audit records for security relevant auditable events.
O.CHANGE_MANAGEMENT	The configuration of, and all changes to, the TOE and its development evidence will be analyzed, tracked, and controlled throughout the TOE’s development.
O.CORRECT_BOOT	The TOE will provide mechanisms to correctly transfer the TSF implementation and TSF data into the TSF’s execution domain.
O.CORRECT_CONFIG	The TOE will provide procedures and mechanisms to generate the TSF configuration data such that the TSF configuration data accurately reflects the user’s intentions regarding partitioning and information flow.
O.CORRECT_LOAD	The TOE will provide procedures and mechanisms to correctly convert the TSF code and/or configuration data into a TSF-useable form.
O.CORRECT_TSF_OPERATION	The TOE will provide a capability to test the TSF to ensure the correct operation of the TSF during normal operation.
O.COVERT_CHANNEL_ANALYSIS	The TOE will undergo appropriate covert channel analysis by NSA to demonstrate that the TOE meets its functional requirement.
O.CRYPTOGRAPHIC_PROTECTION	The TOE will support separation of the cryptography from the rest of the TSF.

O.CRYPTOGRAPHIC_SERVICES	The TOE will use cryptographic mechanisms to protect the integrity of TOE code and data as it resides within the system and when it is transmitted to other systems. The TOE will also use cryptographic mechanisms to verify the integrity of the TSF code and configuration data during initialization. The cryptographic mechanism will use NIST FIPS validated cryptography as a baseline with additional NSA-approved methods.
O.FUNCTIONAL_TESTING	The TOE will undergo independent security functional testing that demonstrates the TSF satisfies the security functional requirements.
O.INSTALL_GUIDANCE	The TOE will be delivered with the appropriate installation guidance to establish and maintain TOE security.
O.INTERNAL_LEAST_PRIVILEGE	The entire TSF will be structured to achieve the principle of least privilege among TSF modules.
O.MANAGE	The TOE will provide all the functions necessary to support the administrative users and authorized subjects in their management of the configuration data, and restrict these functions from use by unauthorized subjects.
O.PROTECT	The TOE will provide mechanisms to protect services and exported resources.
O.RATINGS_MAINTENANCE	Procedures and processes to maintain the TOE’s rating will be documented.
O.RECOVERY	Procedures and/or mechanisms will be provided to assure that recovery, such as from system failure or discontinuity, is obtained without a protection compromise.
O.REFERENCE_MONITOR	The TOE will maintain a domain for its own execution that protects itself and its resources from external interference, tampering, or unauthorized disclosure.
O.RESIDUAL_INFORMATION	The TOE will ensure that any information contained in a protected resource is not released when the resource is reallocated.
O.RESOURCE_SHARING	The TOE will provide mechanisms that mitigate attempts to exhaust TOE resources (e.g., system memory, persistent storage, and processing time).

O.SECURE_STATE	The TOE will provide mechanisms to transition the TSF to a secure state during start-up, re-activation of the current flow policy configuration data and activation of a new flow policy configuration data.
O.SOUND_DESIGN	The TOE will be designed using sound design principles and techniques. The TOE design, design principles and design techniques will be adequately and accurately documented.
O.SOUND_IMPLEMENTATION	The implementation of the TOE will be an accurate instantiation of its design.
O.TRUSTED_DELIVERY	The integrity of the TOE must be protected during the initial delivery and subsequent updates, and verified to ensure that the on-site version matches the master distribution version.
O.TSF_INTEGRITY	The TOE will be able to verify the integrity of the TSF code and data.
O.USER_GUIDANCE	The TOE will provide users with the necessary information for secure use of the TOE.
O.VULNERABILITY_ANALYSIS_TEST	The TOE will undergo independent vulnerability analysis and penetration testing by NSA to demonstrate the design and implementation of the TOE does not allow attackers with high attack potential to violate the TOE's security policies.

4.2 Environment Security Objectives

OE.CHANNELS	If the residual risk from covert channels is a concern, the applications executing on the TOE must be trusted with assurance commensurate with the value of the IT assets protected by the TOE.
OE.PHYSICAL	Physical security will be provided for the TOE by the IT environment commensurate with the value of the IT assets protected by the TOE.

OE.TRUSTED_FLOWS	If a subject is allowed by the configuration data to cause information flow in violation of the partial ordering of information flows between partitions, that subject must be trusted with assurance commensurate with the value of the IT assets in all partitions to which it has access.
OE.TRUSTED_INDIVIDUAL	If an individual is allowed to perform procedures upon which the security of the TOE may depend, that individual is trusted with assurance commensurate with the value of the IT assets.

5. TOE Security Functional Requirement

98 This section contains detailed security functional requirements for the Separation Kernels' trusted security functions (TSF) supporting systems in high robustness environments. The requirements are applied against the Separation Kernel in conjunction with the underlying hardware that supports it. The requirements contained in this section are either selected from Part 2 of the CC or have been explicitly stated (with short names in bold and ending in "_EXP"). Table 5.1 lists the explicit functional requirements in this section.

99 The cryptographic module plays an important role in the enforcement of the TOE security policies. For this reason, the cryptographic related requirements contain more detail than other requirements, in terms of refinements, iterations, and explicitly stated requirements. Refer to section 1.4 to see the notation and formatting used in this profile.

Table 5.1 - Explicit Functional Requirements

Explicit Component	**Component Behavior Name**
FCS_BCM_EXP.1	Baseline Cryptographic Module
FDP_RIP_EXP.2	Full Residual Information Protection
FMT_MSA_EXP.1	Management of Security Attributes
FMT_MTD_EXP.1	Management of TSF Data
FPT_ITI_EXP.1	Inter-TSF Detection of Modification
FPT_TST_EXP.1	TSF Testing
FRU_RSA_EXP.1	Maximum Quotas

5.1 Security Audit (FAU)

5.1.1 Security Audit Automatic Response (FAU_ARP)

5.1.1.1 Security Alarms (FAU_ARP.1)

FAU_ARP.1.1 **Refinement:** The TSF shall take [assignment: *list of the actions* ***to take***] upon detection of **any failure of the TSF self-tests.**[1]

Application Note: The ST author is to fill in the open assignment with the list of actions that are applicable for the TOE's intended use, with particular attention to providing the ability for the TOE to support system-level requirements for fault/failure detection and response.. Acceptable actions may include a visual or audible alarm, a signal/message to the IT environment, or explicit action taken by the TSF (e.g., shutdown).

5.1.2 Security Audit Data Generation (FAU_GEN)

5.1.2.1 Audit Data Generation (FAU_GEN.1)

FAU_GEN.1.1-NIAP-0410 The TSF shall be able to generate an audit record of the following auditable events:

a) Start-up and shutdown of the audit functions;

b) Start-up and shutdown of the TOE;

c) All auditable events **listed in Table 5.2;**

d) [selection: [assignment: *events at a basic level of audit introduced by the inclusion of additional SFRs*], [assignment: *events commensurate with a basic level of audit introduced by the inclusion of explicit requirements*], *no additional events*].

Application Note: There is a broad range of possible auditable events in this category. For example, the nature of these events for a deeply embedded real-time kernel is likely to be that of fault, failure or other exception conditions.

For the first assignment in the selection, the ST author augments the table (or lists explicitly) the audit events associated with the basic level of audit for any SFRs that the ST author includes in the ST that are not included in this PP.

Likewise, for the second assignment the ST author includes audit events that may arise due to the inclusion of any explicit requirements in the ST that are not already in the PP. Because "basic" audit is not defined for such requirements, the ST author will need to determine a set of events that are commensurate with the type of information that is captured at the basic level for similar requirements.

It is acceptable for the ST author to choose "no additional events", if the ST author has not included additional requirements, or has included additional requirements that do not have a basic level (or commensurate level) of audit associated with them. In determining whether or not added functionality should have auditable events, the ST author is to assess the added functionality in terms of its conceptual relationship with the core functionality expressed in this PP and their corresponding requirements for auditable events. As an example: FAU_SEL.1 requires that the set of auditable events be statically determined prior to execution of the TSF and that the set of auditable events are not modifiable during runtime. Since there is no capability to modify the audit behavior at runtime, there is no requirement to audit changes to the runtime behavior. However, should the ST author provide the capability for authorized subjects to modify the behavior of the audit mechanism during runtime, then any such runtime modification constitutes an auditable event.

Application Note: The TSF is not required to generate a structured audit "record" in any specified format. The TSF is expected to capture data that identifies and characterizes the event as defined in Table 5-2 Auditable Events, and provide a capability for the IT environment to "pull" that information from the TSF. The TSF is not required to notify the IT environment of the existence of the audit data and the TSF is not required to "push" the information to the IT environment.

Application Note: It is common that for purposes of engineering analysis, real-time embedded systems record operational data to support analysis and debugging. This data is referred to as instrumentation. This data is not necessarily security relevant, that is, not associated with enforcement of the security policy. The audit data generation requirements should not to be *confused with instrumentation requirements levied by applications assuming the instrumentation requirements do not violate the minimization requirements.*

Table 5.2 Auditable Events

Requirement	**Audit events prompted by requirement**
Security Alarms (FAU_ARP.1)	• Actions taken due to failure of TSF self tests
Audit Data Generation (FAU_GEN.1)	(None)
Selective Audit (FAU_SEL.1)	(None)
Explicit: Baseline Cryptographic Module (FCS_BCM_EXP.1)	(None)
Complete Information Flow Control (for Information Flow Control Policy) (FDP_IFC.2(1))	(None)
Complete Information Flow Control (for Partition Flow Control Policy) (FDP_IFC.2(2))	(None)
Simple Security Attributes (FDP_IFF.1)	• Establishment of connections between partitions. *Application Note: The TSF is not required to audit each instance of an information flow between partitions. The TSF is required to provide the capability to audit the establishment of each connection between partitions.*
Partial Elimination of Illicit Information Flows (FDP_IFF.4)	• The use of identified illicit information flow channels.
Full Residual Information Protection (FDP_RIP_EXP.2)	(None)
Explicit: Management of Security Attributes (FMT_MSA_EXP.1)	(None)
Explicit: Management of TSF Data (for Modification of Flow Policy Configuration Data) (FMT_MTD_EXP.1)	• (None)
Secure TSF Data (FMT_MTD.3)	• All rejected values of TSF data.
Underlying Abstract Machine Test (FPT_AMT.1)	• Failures detected by tests of the underlying abstract machine and the results of the tests
Fail Secure (FPT_FLS.1)	• Failures detected by the FPT_AMT.1 and FMT_TST.1 tests.
Explicit: Inter-TSF Detection of Modification (FPT_ITI_EXP.1)	• The detection of modification of transmitted TSF data. • The action taken upon detection of modification of transmitted TSF data.

Automated Recovery (FPT_RCV.2)	• The fact that a failure or service discontinuity occurred. • Resumption of the regular operation. • Type of failure or service discontinuity.
Function Recovery (FPT_RCV.4)	• If possible, the impossibility to return to a secure state after failure or a security function. • If possible, the detection of a failure of a security function.
Non-Bypassability of the TSF (FPT_RVM.1)	(None)
Complete Reference Monitor (FPT_SEP.3)	(None)
Reliable Time Stamp (FPT_STM.1)	• (None)
Explicit: TSF Testing (FPT_TST_EXP.1)	• Failures of TSF self tests and the results of the tests.
Explicit: Maximum Quotas (for System Memory) (FRU_RSA.1(1))	• Rejection of allocation operation due to system memory limits.
Explicit: Maximum Quotas (for Processing Time) (FRU_RSA.1(2))	• Rejection of allocation operation due to processing time limits.

FAU_GEN.1.2-NIAP-0410 The TSF shall record within each audit record at least the following information:

a) Date and time of the event, type of event, subject identity, and the outcome (success or failure) of the event; and

b) For each audit event type, based on the auditable event definitions of the functional components included in the PP/ST,

• the identity of the resource;

Application Note: The TSF is not required to explicitly provide date and time of the event. It is acceptable for the TSF to provide a timestamp that reflects relative time within the TSF so long as the IT environment is able to correlate that timestamp to date and time and the IT environment is able to establish event sequences based upon timestamp values.

Application Note: Other audit relevant information associated with security-relevant functions that are included in the ST but that are not included in this PP should be included within the audit record.

5.1.3 Security Audit Event Selection (FAU_SEL)

5.1.3.1 Selective Audit (FAU_SEL.1)

FAU_SEL.1.1 The TSF shall be able to include or exclude auditable events from the set of audited events based on the following attributes:

a) resource identity,

b) subject identity,

c) event type,

d) **success of auditable security events,**

e) **failure of auditable security events,**

f) **[selection: [assignment: list of additional attributes specific to the audit capabilities of the implementation], no additional attributes].**

Application Note: The TSF is not expected to provide a run-time capability for management of the audit function behavior. It is acceptable for the TSF to provide the means for the audit function behavior to be specified by the configuration data, and for that behavior to be static and remain in effect until such time that the TSF is initialized with a different set of audit configuration data.

5.2 Cryptographic Support (FCS)

5.2.1 Explicit: Baseline Cryptographic Module (FCS_BCM_EXP)

5.2.1.1 Explicit: Baseline Cryptographic Module (FCS_BCM_EXP.1)

FCS_BCM_EXP.1.1 All cryptographic modules shall comply with FIPS PUB 140-2 when performing FIPS-approved cryptographic functions in FIPS-approved cryptographic modes of operation.

Application Note: This requirement is met by presenting a NIST FIPS PUB 140-2 certificate.

FCS_BCM_EXP.1.2 Cryptographic functions and cryptographic modes of operation as identified in this PP shall be NSA-validated.

FCS_BCM_EXP.1.3 All cryptographic modules implemented in the TSF ***[selection:***

(1) Entirely in hardware shall comply with, at a minimum, FIPS PUB 140-2, Level 3. For Cryptographic Key Management, only Key Entry and Key Storage functions are required;

(2) Entirely in software shall comply with, at a minimum, FIPS PUB 140-2, Level 1 and also FIPS PUB 140-2, Level 3 for the following: Cryptographic Module Ports and Interfaces; Cryptographic Key Management; and Design Assurance. For Cryptographic Key Management, only Key Entry and Key Storage functions are required;

(3) As a combination of hardware and software shall comply with, at a minimum, FIPS PUB 140-2, Level 1 excluding and also FIPS PUB 140-2, Level 3 for the following: Cryptographic Module Ports and Interfaces; Cryptographic Key Management; and Design Assurance. For Cryptographic Key Management, only the Key

Entry and Key Storage functions are required.]

Application Note: "Roles, Services and Authentication" is excluded because there are no Identification & Authentication requirements in this SKPP.

Application Note: "Combination of hardware and software" means that some part of the cryptographic functionality will be implemented as a software component of the TSF. The combination of a cryptographic hardware module and a software device driver whose sole purpose is to communicate with the hardware module is considered a hardware module rather than a "combination of hardware and software".

5.3 User Data Protection (FDP)

5.3.1 Information Flow Control Policy (FDP_IFC)

5.3.1.1 Complete Information Flow Control (for Information Flow Control Policy) (FDP_IFC.2(1))

FDP_IFC.2.1(1) **Refinement**: The TSF shall enforce the **Information Flow Control policy** on **all subjects, all exported resources** and all operations that cause information to flow to and from subjects.**4**

FDP_IFC.2.2(1) **Refinement**: The TSF shall ensure that all operations that cause any information to flow to and from any subject are covered by **the Information Flow Control** SFP.**5**

Application Note: This information control policy applies to all subjects. The subjects are defined as an active entity from the perspective of the TSF. An acceptable implementation may allocate, at most, only one subject per partition or could allocate multiple subjects per partition. Note that the configuration data could specify a configuration where a partition has no subject allocated to it.

*As with subjects, this policy applies to *all* exported resources available at the TSFI. The granularity of the policy enforcement should be at the granularity of the exported resources. An acceptable implementation may provide very abstract representation of resources at the TSFI, such as communication channels, stacks, and semaphores. Another acceptable implementation may provide resources to the granularity of memory blocks, hardware registers, hardware devices, etc. The TOE-specific policy needs to apply to all exported resources.*

5.3.1.2 Complete Information Flow Control (for Partition Flow Control Policy) (FDP_IFC.2(2))

FDP_IFC.2.1(2) **Refinement**: The TSF shall enforce the **Partition Flow Control policy** on **all flows between partitions** and all operations that cause information to flow to and from **partitions**. **6**

FDP_IFC.2.2(2) **Refinement**: The TSF shall ensure that all operations that cause any information to flow to and from any **partition** are covered by **the Partition Flow Control** SFP. **7**

5.3.2 Information Flow Control Functions (FDP_IFF)

5.3.2.1 Simple Security Attributes (FDP_IFF.1)

FDP_IFF.1.1-NIAP-0407 **Refinement**: The TSF shall enforce the **Information Flow Control policy and the Partition Flow Control policy** based on the following types of security attributes: **8**

a) **The identity of a subject;**

b) **The identity of a resource;**

c) **The mode of operation;**

d) **The identity of the single partition to which the subject is assigned;**

e) **The identity of the single partition to which the resource is assigned.**

Application Note: It is acceptable for a subject ID or a resource ID to be the same as its partition ID if the subject or resource is in a partition that includes no other subject or resource, thus obviating the need to explicitly assign that subject or resource to a partition. Such a subject or resource ID may be considered to be a partition ID for the purposes of enforcing FDP_IFF.1.2.a.

FDP_IFF.1.2-NIAP-0407 **Refinement**: The TSF shall permit an information flow between a subject and a **resource** via a controlled operation if the following rules hold: **9**

a) **The mode of operation is allowed by the configuration data for the partition to which the identified subject is assigned and the partition to which the identified resource is assigned; and**

b) **The mode of operation is allowed by the configuration data for the identified subject and the identified resource.**

Application Note: For FDP_IFF.1.2, implicit forms of allowance, such as regular expressions or defaults are acceptable implementations.

Application Note: For flows between subjects and resources that have IDs that are the same as their respective partition IDs, the TSF need not maintain separate configuration information regarding rules a and b.

FDP_IFF.1.3-NIAP-0407 **Refinement**: The TSF shall enforce the following **separation** rule: **If the mode of operation is not explicitly allowed, per FDP_IFF.1.2, then deny the requested information flow. 10**

FDP_IFF.1.4-NIAP-0407 The TSF shall provide <u>no additional SFP capabilities</u>.

FDP_IFF.1.5-NIAP-0407 The TSF shall explicitly authorize an information flow based on the following rules: <u>no explicit authorization rules</u>.

FDP_IFF.1.6-NIAP-0407 The TSF shall explicitly deny an information flow based on the following rules: <u>no explicit denial rules.</u>

5.3.2.2 Partial Elimination of Illicit Information Flows (FDP_IFF.4)

FDP_IFF.4.1 The TSF shall enforce the **Partition Flow Control policy** to limit the capacity of **covert timing channels between partitions** to a [assignment: *maximum capacity*].

FDP_IFF.4.2 **Refinement**: The TSF shall **enforce the Partition Flow Control policy to** prevent **covert storage channels between partitions**.11

5.3.3 Residual Information Protection (FDP_RIP)

5.3.3.1 Explicit: Full Residual Information Protection (FDP_RIP_EXP.2)

FDP_RIP_EXP.2.1 The TSF shall ensure that any previous information content is made unavailable upon the [selection: *allocation, deallocation*] of all exported resources.

5.4 Identification and Authentication (FIA)

100 5.4.1 Subject and Resource Attribute Definition (FIA_ATD)

5.4.1.1 Explicit: Subject and Resource Attribute Definition (FIA_ATD_EXP.1)

FIA_ATD_EXP.1.1 For each subject, the TSF configuration data shall include the following list of security attributes: [assignment: *list of subject security attributes*].

FIA_ATD_EXP.1.2 For each resource, the TSF configuration data shall include the following list of security attributes: [assignment: *list of resource security attributes*].

Application Note: The configuration data fulfills a function that is equivalent to what is traditionally performed by an authorized individual with the responsibility for defining users and granting authorization for users to interact with objects. For the separation kernel, that function is expanded to include defining resource attributes as the configuration data must explicitly define all resources.

5.4.2 Subject and Resource Attribute Binding (FIA_USB)

5.4.2.1 Explicit: Subject and Resource Attribute Binding (FIA_USB_EXP.1.1)

FIA_USB_EXP.1.1 Upon initialization of the TSF, the TSF shall associate the internal representation of all security attributes defined for each subject with the subject as specified in the configuration data.

FIA_USB_EXP.1.2 Upon initialization of the TSF, the TSF shall associate the internal representation of all security attributes defined for each resource with the resource as specified in the configuration data.

Application Note: The concept of user-subject binding applies to the separation kernel in the sense that *the TSF is required to perform the binding of subject and resource attributes defined in the configuration data to the internal representation of those attributes for each subject and resource when it is created and assigned to a partition.*

5.5 Security Management (FMT)

Application Note: The concept of security management for a separation kernel differs from that of general security kernel since it is not anticipated that there will be direct interaction with the separation kernel by authorized individuals such as system administrators. It is more likely that security management of the separation kernel will be accomplished exclusively via the pre-runtime static definition of the separation kernel capabilities, exclusively via the dynamic runtime management of the separation kernel capabilities by authorized application processes (such as a system controller application), or via a combination of the two.

This protection profile has taken the former approach and assumes that the configuration data completely defines the separation kernel runtime configuration, and that configuration remains static for the duration of the execution of the separation kernel. As a result, there are no FMT requirements for authorized subjects to interact with the TSF for the purpose of managing the TSF.

It is appropriate for the Security Target author to express security relevant management capabilities that are specific to and consistent with the intended operational environment in which the separation kernel will be placed and the needs of middleware and application level processes with responsibility to manage the separation kernel.

Should this option be exercised, then the Security Target author should use this FMT section to state the management requirements for granting authorization to subjects and determining what authorized subjects can and can not do in regards to invoking TSF functions and accessing TSF data. Additionally, the ST author must update the audit requirements in the FAU section to reflect the auditable events associated with providing runtime management capabilities to some authorized subjects while restricting the use of those capabilities on others.

5.5.1 Management of Security Attributes (FMT_MSA)

5.5.1.1 Explicit: Management of Security Attributes (FMT_MSA_EXP.1)

FMT_MSA_EXP.1.1 The TSF shall restrict the ability to assign authorized subjects to the configuration data.

Application Note: This requires the configuration data to be the only means through which subjects are designated as "authorized subjects". During initialization, the TSF assigns authorizations to subjects as specified by the configuration data.

5.5.2 Management of TSF Data (FMT_MTD)

5.5.2.1 Explicit: Management of TSF Data (for Modification of Flow Policy Configuration Data) (FMT_MTD_EXP.1)

FMT_MTD_EXP.1.1 The TSF shall prevent modification of the flow policy configuration data.

5.5.2.2 Secure TSF Data (FMT_MTD.3)

FMT_MTD.3.1 Refinement: The TSF shall ensure that only **valid** values are accepted for TSF data. **13**

Application Note: Valid implies that the values fall within the defined range for the TSF data (e.g., an audit enable/disable indicator must be within range of a Boolean type).

5.6 Protection of the TSF (FPT)

5.6.1 Underlying Abstract Machine Test (FPT_AMT)

5.6.1.1 Abstract Machine Testing (FPT_AMT.1)

FPT_AMT.1.1 **Refinement:** The TSF shall run a suite of tests during the initial start-up, during automated recovery and periodically during normal operation to demonstrate the correct operation of the security assumptions provided by the abstract machine that underlies the **software portions of the** TSF.

Application Note: The test suite need only cover aspects of the underlying abstract machine on which the TSF relies to implement required functions, including domain separation.

Application Note: The test suite for periodic testing may be a subset of the initial start-up test. The period test suite should be a maximal set of tests that can be run without interfering with the normal system operation. The period test suite may be further divided into different test groups. Each test group may be scheduled to run at different time during run-time.

Application Note: Annex J of the CC, Part 2, explains that with respect to the FPT class, the TSF consists of three parts: a) the TSF's abstract machine, b) the TSF's implementation, and c) the TSF data. This component covers the testing of the TSF's abstract machine which is defined in Annex J as "the virtual or physical machine upon which the specific TSF implementation under evaluation executes."

5.6.2 Explicit: Establishment of Secure State (FPT_ESS)

5.6.2.1 Explicit: Establishment of Secure State (FPT_ESS_EXP.1)

FPT_ESS_EXP.1.1 The TSF shall ensure secure state upon activation of any partition or information flow policy.

Application Note: The phrase "upon activation of any partition or information flow policy" means that the TSF has been initialized (i.e., the non-TSF trusted initialization functions have successfully completed), the TSF has completed the subject-attribute to subject binding and resource-attribute to resource binding, and the TSF is now ready to sustain secure runtime operations.

5.6.3 Fail Secure (FPT_FLS)

5.6.3.1 Failure with Preservation of Secure State (FPT_FLS.1)

FPT_FLS.1.1 The TSF shall preserve a secure state when the following types of failures occur: [assignment: *list of failures that are detected by tests defined in FPT_AMT.1 and FPT_TST.1 and that require preservation of* *secure state*].

Application Note: The ST author is to provide the list of post-initialization failures that can be detected and for which the TSF can respond to and preserve a secure state.

Application Note: TSF failure modes vary and may include "hard" failures such as those associated with hardware failure or unrecoverable software errors, and "soft" failures such as intermittent hardware errors and recoverable software errors.

The TSF is not expected to protect itself against all types of hardware errors. For example, a radiation induced change of a single bit in a memory access control register could result in an incorrect (but valid) memory location being accessed. This would not be detected by the hardware.

5.6.4 Integrity of Exported TSF Data (FPT_ITI)

5.6.4.1 Explicit: Inter-TSF Detection of Modification (FPT_ITI_EXP.1)

FPT_ITI_EXP.1.1 The TSF shall provide the capability to detect modification of all TSF data whenever the TSF data is transmitted between the TSF and a remote trusted IT product within the following metric: [assignment: *a defined modification metric*].

FPT_ITI_EXP.1.2 The TSF shall provide the capability to verify the integrity of all TSF data whenever the TSF data is transmitted between the TSF and a remote trusted IT product and perform [assignment: *action to be taken*] if modifications are detected.

Application Note: This requirement applies to TSF data such as the configuration data for those cases where the TSF obtains the configuration data from a remote trusted product in the IT environment. .

5.6.5 Trusted Recovery (FPT_RCV)

5.6.5.1 Automated Recovery (FPT_RCV.2)

FPT_RCV.2.1 Refinement: When automated recovery from a failure or service discontinuity is not possible, the TSF shall enter a maintenance mode where the ability to return the TOE to an operational secure state is provided.

Application Note: The word "operational" has been inserted to make it clear that the desired secure state is a secure state in operational mode as opposed to a secure state in maintenance mode. See the Glossary of Terms section for the description of maintenance mode and operational mode.

Application Note: There is no requirement that the TSF supports the recovery action to transition from the maintenance mode to the operational mode. Entrance to maintenance mode can be achieved by halting the system, and return to a secure state can be achieved by system start-up.

FPT_RCV.2.2 Refinement: The TSF shall ensure the return of the TOE to a operational secure state using automated procedures for the following failures/service discontinuities:

a) Power failures,

b) [selection: [assignment: *list of additional failures/service discontinuities], no other failures/service discontinuities*].

Application Note: It is assumed that all systems can at least recover automatically from a power failure.

5.6.5.2 Function Recovery (FPT_RCV.4)

FPT_RCV.4.1 The TSF shall ensure that **all SFs that affect the secure state and** [assignment: *list of failure scenarios*] have the property that the SF either completes successfully, or for the indicated failure scenarios, recovers to a consistent and secure state.

5.6.6 Reference Mediation (FPT_RVM)

5.6.6.1 Non-Bypassability of the TSP (FPT_RVM.1)

FPT_RVM.1.1 The TSF shall ensure that TSP enforcement functions are invoked and succeed before each function within the TSC is allowed to proceed.

5.6.7 Domain Separation (FPT_SEP)

5.6.7.1 Complete Reference Monitor (FPT_SEP.3)

FPT_SEP.3.1 The unisolated portion of the TSF shall maintain a security domain for its own execution that protects it from interference and tampering by untrusted subjects.

FPT_SEP.3.2 The TSF shall enforce separation between the security domains of subjects in the TSC.

FPT_SEP.3.3 **Refinement**: The TSF shall maintain the part of the TSF that enforces the information flow control, **partition flow control and cryptography** SFPs in a security domain for its own execution that protects them from interference and tampering by the remainder of the TSF and by subjects untrusted with respect to the TSP. **14**

Application Note: Ideally, use of off board hardware or a third processor hardware state is the most preferred implementation supporting separation, because it would protect the part of the TSF that enforces the cryptography from all other parts of the TSF. Migration to this most preferred implementation is anticipated eventually.

Application Note: For separation kernels, it is not required to separate the unisolated portion of the TSF from the part of the TSF that enforces the flow control and cryptography SFPs with hardware mechanisms. Software layering is sufficient to meet this requirement.

5.6.8 Time Stamps (FPT_STM)

5.6.8.1 Reliable Time Stamp (FPT_STM.1)

FPT_STM.1.1 The TSF shall be able to provide reliable time stamps for its own use.

Application Note: It is the responsibility of the ST developer to provide a definition and metric for the term "reliable time stamp" and to provide evidence that the implementation meets the defined definition and metric. The rational in the ST should be used to substantiate the chosen definition and metric.

Application Note: Whatever native format the system keeps time in is acceptable. For example, a monotonically increasing counter with a defined metric for each increment of the counter is an acceptable implementation of this requirement.

5.6.9 TSF Self Test (FPT_TST)

5.6.9.1 Explicit: TSF Testing (FPT_TST_EXP.1)

FPT_TST_EXP.1.1 The TSF shall run a suite of self tests during the initial start-up, during automated recovery and [assignment: *conditions under which self test should occur during normal operation*] to demonstrate the correct operation of the TSF's implementation.

Application Note: See Annex J of the CC, Part 2, for an explanation of the notion of TSF's implementation.

FPT_TST_EXP.1.2 The TSF shall verify, or provide the capability for an authorized subject to verify, the integrity of TSF configuration data and [assignment: list of *additional TSF data upon which the TSF depends to enforce its security policies correctly*].

FPT_TST_EXP.1.3 The TSF shall verify, or provide the capability for an authorized subject to verify, the integrity of stored TSF executable code.

5.7 Resource Utilization (FRU)

5.7.1 Resource Allocation (FRU_RSA)

Application Note: If the total allocation exceeds the maximum limit of physical resources, potential covert channels between partitions could be introduced.

5.7.1.1 Explicit: Maximum Quotas (for System Memory) (FRU_RSA_EXP.1(1))

FRU_RSA_EXP.1.1(1) The TSF shall enforce the allocation of system memory that partitions can use simultaneously as defined by the configuration data.

5.7.1.2 Explicit: Maximum Quotas (for Processing Time) (FRU_RSA_EXP.1(2))

FRU_RSA_EXP.1.1(2) The TSF shall enforce the allocation of processing time that partitions can use over a specified period of time as defined by the configuration data.

Application Note: The algorithm to determine percentages of time can be based on many factors (e.g., number of partitions, relative priority of partitions, availability of resources to partitions).

End Notes

This section records the functional requirements where deletions of Common Criteria text were performed.

1 A modification of CC text was performed in FAU_ARP.1.1. The words "least disruptive actions" were replaced by the words "actions to take" and the words "a potential security violation" were replaced with "any failure of the TSF self-tests".

FAU_ARP.1.1 Refinement: The TSF shall take [assignment: list of the ~~least disruptive~~ actions **to take**] upon detection of ~~a potential security violation~~ **any failure of the TSF self-tests**.

3 A deletion of CC text was performed in FCS_COP.1.1(2). Rationale: The words "a specified cryptographic" were deleted for clarity and better flow of the requirement.

FCS_COP.1.1(2) - **Refinement**: The TSF shall perform **cryptographic signature services** in accordance with ~~a specified cryptographic~~ the **NIST-approved digital signature** algorithm ***[selection:***

4 A deletion of CC text was performed in FDP_IFC.2.1(1). Rationale: The words "covered by the SFP" were deleted to remove the possible implication that there may be some subjects that are not covered by the Information Flow Control SFP.

FDP_IFC.2.1(1) **Refinement**: The TSF shall enforce the **Information Flow Control policy** on **all subjects**, **all exported resources** and all operations that cause information to flow to and from subjects ~~covered by the SFP~~.

5 A deletion of CC text was performed in FDP_IFC.2.2(1). Rationale: The words "in the TSC" were deleted to remove the possible implication that there may be information or subjects outside the TSC. The words "an information flow control" were changed to "the Information Flow Control" to clarify that this requirement applies only to the Information Flow Control SFP.

FDP_IFC.2.2(1) **Refinement**: The TSF shall ensure that all operations that cause any information ~~in the TSC~~ to flow to and from any subject ~~in the TSC~~ are covered by ~~an information flow control~~ **the Information Flow Control** SFP.

6 A deletion of CC text was performed in FDP_IFC.2.1(2). Rationale: The open assignment variables "list of subjects and information" were assigned to "all flows between partitions" and the word "subjects" was changed to "partitions" to narrow this requirement to information flows between partitions. The words "covered by the SFP" were deleted to remove the possible implication that there may be some partitions that are not covered by the Partition Flow Control SFP.

FDP_IFC.2.1(2) **Refinement**: The TSF shall enforce the **Partition Flow Control policy** on ~~[assignment: lists of subjects and information] and~~ **all flows between partitions** and all operations that cause information to flow to and from ~~subjects~~ **partitions** ~~covered by the SFP~~.

7 A deletion of CC text was performed in FDP_IFC.2.2(2). Rationale: The words "in the TSC" were deleted to remove the possible implication that there may be information or partitions outside the TSC. The words "subject" and "an information flow control" were changed to clarify that this requirement applies only to the Partition Flow Control SFP.

FDP_IFC.2.2(2) **Refinement**: The TSF shall ensure that all operations that cause any information ~~in the TSC~~ to flow to and from any ~~subject~~ **partition** ~~in the TSC~~ are covered by the Partition Flow Control SFP.

8 A deletion of CC text was performed in FDP_IFF.1.1-NIAP-0407. Rationale: The words "subject and information" were deleted to refine the scope of the security attributes.

FDP_IFF.1.1-NIAP-0407 **Refinement**: The TSF shall enforce the **Information Flow Control policy and the Partition Flow Control policy** based on the following types of ~~subject and information~~ security attributes:

9 A deletion of CC text was performed in FDP_IFF.1.2-NIAP-0407. Rationale: The word "controlled" was deleted as a modifier to "subject" since all subjects are controlled; there are no uncontrolled subjects. The word "controlled" was deleted as a modifier to "resource," since all resources are controlled; there are no uncontrolled resources. "Information" was changed to "resource" to encompass both information and aspects of the underlying hardware.

FDP_IFF.1.2-NIAP-0407 **Refinement**: The TSF shall permit an information flow between a ~~controlled~~ subject and ~~controlled information~~ a resource via a controlled operation if the following rules hold:

10 A deletion of CC text was performed in FDP_IFF.1.3-NIAP-0407. Rationale: The words "information flow control" were deleted to narrow the scope of the applicable rules.

FDP_IFF.1.3-NIAP-0407 **Refinement**: The TSF shall enforce the following ~~information flow control~~ **separation** rule: **If the mode of operation is not explicitly allowed, per FDP_IFF.1.2, then deny the requested information flow**.

11 A modification of CC text was performed in FDP_IFF.4.2. Rationale: The words "enforce the Partition Flow Control policy to" were added to clarify that this requirement only applies to covert storage channels between partitions.

FDP_IFF.4.2 **Refinement**: The TSF shall **enforce the Partition Flow Control policy to** prevent **covert storage channels between partitions**.

13 A modification of CC text was performed in FMT_MTD.3.1. Rationale: The word "secure" was changed to "valid" to indicate that this is intended to be a syntax check. The words "other than security attributes" were added to clarify the types of TSF data that are covered by this requirement.

FMT_MTD.3.1 Refinement: The TSF shall ensure that only ~~secure~~ **valid** values are accepted for TSF data **other than security attributes**.

14 A modification of CC text was performed in FPT_SEP.3.3. Rationale: The words "access control and/or" were deleted since the TSF does not enforce any access control SFPs. The words "partition flow control and cryptography" were added to clarify the types of SFPs that are required to be enforced in a domain distinct from the remainder of the TSF.

FPT_SEP.3.3 **Refinement:** The TSF shall maintain the part of the TSF that enforces the ~~access control and/or~~ information flow control, **partition flow control and cryptography** SFPs in a security domain for its own execution that protects them from interference and tampering by the remainder of the TSF and by subjects untrusted with respect to the TSP.

6. TOE Security Assurance Requirements

101 This section contains the detailed security assurance requirements for Separation Kernels supporting systems in environments requiring high robustness. The requirements contained in this section are either selected from Part 3 of the CC or have been explicitly stated (with short names ending in "_EXP"). Table 6.1 lists the explicitly stated assurance components.

Table 6.1 - Explicit Assurance Requirements

Explicit Component	Component Behavior Name
ADO_DEL_EXP.2	Detection of Modification
ADV_ARC_EXP.1	Architectural Design
ADV_CMP_EXP.2	Detailed Composition Information
ADV_FSP_EXP.6	Formal Functional Specification With Indirect Error Mapping
ADV_HDW_EXP.1	Development Requirements for Hardware (TBD)
ADV_HLD_EXP.4	Semiformal High Level Design
ADV_IFA_EXP.1	Availability of Interface Information
ADV_IMP_EXP.3	Verified Implementation of the TSF
ADV_INI_EXP.1	Trusted Initialization (TBD)
ADV_INT_EXP.4	Minimization of Complexity
ADV_LLD_EXP.4	Semiformal Low Level Design
ADV_RCR_EXP.3	Formal Correspondence Demonstration
ADV_SPM_EXP.3	Formal TOE Security Policy Model
AGD_ADM_EXP.1	Administrator Guidance
AMA_AMP_EXP.1	Assurance Maintenance Plan

102 The combination of assurance components is equivalent to an Evaluated Assurance Level 6 with augmentation (EAL6+). The augmented assurances required are in the areas of development, independent testing, systematic flaw remediation and maintenance of assurance. The intended TOE environment and the value of information processed by this environment establish the need for the TOE to be evaluated at this EAL level[5]. These security assurance requirements are summarized in Table 6.2. Note that flaw remediation (ALC_FLR) and maintenance of assurance (AMA_AMP_EXP) have also been chosen even though the CC chose not to assign these components to a specific EAL level.

[5] Refer to the "Mutual Recognition of Common Criteria Certificates" section 1.3 to read conditions for the CC certificate to be mutually recognized for PPs with EALs higher than 4.

103 With respect to development requirements in the ADV classes, a structured document such as the Common Criteria is sufficient to meet the requirements for semiformal specification.

Table 6.2 - Summary of Assurance Components by Evaluation Assurance Level

Assurance Class	Assurance Family	Assurance Components by Evaluation Assurance Level						
		EAL1	EAL2	EAL3	EAL4	EAL5	**EAL6**	EAL7
Configuration Management	ACM_AUT				1	1	**2**	2
	ACM_CAP	1	2	3	4	4	**5**	5
	ACM_SCP			1	2	3	**3**	3
Delivery and Operation	ADO_DEL_EXP		1	1	2	2	(2)	3
	ADO_IGS	1	1	1	1	1	**1**	1
Development (TSF)	ADV_ARC_EXP				(1)	(1)	**(1)**	(1)
	ADV_CMP_EXP		(1)	(2)	(2)	(2)	**(2)**	(2)
	ADV_FSP_EXP	(1)	(2)	(3)	(3)	(4)	(5)	**(6)**
	ADV_HLD_EXP		(1)	(2)	(2)	(3)	**(4)**	(4)
	ADV_IFA_EXP						**(1)**	
	ADV_IMP_EXP				(1)	(1)	(2)	**(3)**
	ADV_INT_EXP					(2)	(3)	**(4)**
	ADV_LLD_EXP				(1)	(2)	(3)	**(4)**
	ADV_RCR		(1)	(1)	(2)	(2)	(2)	**(3)**
	ADV_SPM				(1)	(3)	**(3)**	(3)
Guidance Documents	AGD_ADM_EXP	1	1	1	1	1	**(1)**	1
	AGD_USR	1	1	1	1	1	**1**	1
Life cycle Support	ALC_DVS			1	1	1	**2**	2
	ALC_FLR						**(3)**	
	ALC_LCD				1	2	**2**	3
	ALC_TAT				1	2	**3**	3
Maintenance of Assurance	AMA_AMP_EXP						**(1)**	
Tests	ATE_COV		1	2	2	2	**3**	3
	ATE_DPT			1	1	2	**2**	3
	ATE_FUN		1	1	1	1	**2**	2
	ATE_IND	1	2	2	2	2	2	**3**
Vulnerability Assessment	AVA_CCA_EXP					1	**(2)**	2
	AVA_MSU			1	2	2	**3**	3
	AVA_SOF		1	1	1	1	**1**	1
	AVA_VLA		1	1	2	3	**4**	4

6.1 Configuration Management (ACM)

6.1.1 CM Automation (ACM_AUT)

6.1.1.1 Complete CM Automation (ACM_AUT.2)

ACM_AUT.2.1D The developer shall use a CM system.

ACM_AUT.2.2D The developer shall provide a CM plan.

ACM_AUT.2.1C The CM system shall provide an automated means by which only authorized changes are made to the TOE implementation representation, and to all other configuration items.

ACM_AUT.2.2C The CM system shall provide an automated means to support the generation of the TOE.

ACM_AUT.2.3C The CM plan shall describe the automated tools used in the CM system.

ACM_AUT.2.4C The CM plan shall describe how the automated tools are used in the CM system.

ACM_AUT.2.5C The CM system shall provide an automated means to ascertain the changes between the TOE and its preceding version.

ACM_AUT.2.6C The CM system shall provide an automated means to identify all other configuration items that are affected by the modification of a given configuration item.

ACM_AUT.2.1E The evaluator shall confirm that the information provided meet all requirements for content and presentation of evidence.

6.1.2 CM Capabilities (ACM_CAP)

6.1.2.1 Advanced Support (ACM_CAP.5)

ACM_CAP.5.1D The developer shall provide a reference for the TOE.

ACM_CAP.5.2D The developer shall use a CM system.

ACM_CAP.5.3D The developer shall provide CM documentation.

ACM_CAP.5.1C The reference for the TOE shall be unique to each version of the TOE.

ACM_CAP.5.2C The TOE shall be labeled with its reference.

ACM_CAP.5.3C The CM documentation shall include a configuration list, a CM plan, an acceptance plan, and integration procedures.

ACM_CAP.5.4C The configuration list shall describe the configuration items that comprise the TOE.

ACM_CAP.5.5C The CM documentation shall describe the method used to uniquely identify the configuration items.

ACM_CAP.5.6C The CM system shall uniquely identify all configuration items.

ACM_CAP.5.7C The CM plan shall describe how the CM system is used.

ACM_CAP.5.8C The evidence shall demonstrate that the CM system is operating in accordance with the CM plan.

ACM_CAP.5.9C The CM documentation shall provide evidence that all configuration items have been and are being effectively maintained under the CM system.

ACM_CAP.5.10C The CM system shall provide measures such that only authorized changes are made to the configuration items.

ACM_CAP.5.11C The CM system shall support the generation of the TOE.

ACM_CAP.5.12C The acceptance plan shall describe the procedures used to accept modified or newly created configuration items as part of the TOE.

ACM_CAP.5.13C The integration procedures shall describe how the CM system is applied in the TOE manufacturing process.

ACM_CAP.5.14C The CM system shall require that the person responsible for accepting a configuration item into CM is not the person who developed it.

ACM_CAP.5.15C The CM system shall clearly identify the configuration items that comprise the TSF.

ACM_CAP.5.16C The CM system shall support the audit of all modifications to the TOE, including as a minimum the originator, date, and time in the audit trail.

ACM_CAP.5.17C The CM system shall be able to identify the master copy of all material used to generate the TOE.

ACM_CAP.5.18C The CM documentation shall demonstrate that the use of the CM system, together with the development security measures, allow only authorized changes to be made to the TOE.

ACM_CAP.5.19C The CM documentation shall demonstrate that the use of the integration procedures ensures that the generation of the TOE is correctly performed in an authorized manner.

ACM_CAP.5.20C The CM documentation shall demonstrate that the CM system is sufficient to ensure that the person responsible for accepting a configuration item into CM is not the person who developed it.

ACM_CAP.5.21C The CM documentation shall justify that the acceptance procedures provide for an adequate and appropriate review of changes to all configuration items.

ACM_CAP.5.1E The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

6.1.3 CM Scope (ACM_SCP)

6.1.3.1 Development Tools CM Coverage (ACM_SCP.3)

ACM_SCP.3.1D The developer shall provide CM documentation.

ACM_SCP.3.1C The CM documentation shall show that the CM system, as a minimum, tracks the following: the TOE implementation representation, design documentation, test documentation, user documentation, administrator documentation, CM documentation, security flaws, and development tools and related information.

ACM_SCP.3.2C The CM documentation shall describe how configuration items are tracked by the CM system.

ACM_SCP.3.1E The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

6.2 Delivery and Operation (ADO)

6.2.1 Delivery (ADO_DEL)

6.2.1.1 Explicit: Detection of Modification (ADO_DEL_EXP.2)

ADO_DEL_EXP.2D The developer shall document procedures for delivery of the TOE or parts of it to the user.

ADO_DEL_EXP.2.2D The developer shall use the delivery procedures.

ADO_DEL_EXP.2.3D The developer shall use independent channels to deliver the TOE code and to deliver the cryptographic keying materials used to verify the delivery of the code.

ADO_DEL_EXP.2.4D The developer shall use cryptographic signature services in accordance with the NIST-approved digital signature algorithm *[selection:*

(1) Digital Signature Algorithm (DSA) with a key size (modulus) of 2048 bits or greater,

(2) RSA Digital Signature Algorithm (rDSA with odd e) with a key size (modulus) of 2048 bits or greater, or

(3) Elliptic Curve Digital Signature Algorithm (ECDSA) with a key size of 256 bits or greater]

that meets the following:3

a) Case: Digital Signature Algorithm

FIPS PUB 186-2[6], Digital Signature Standard, for signature creation and verification processing; and ANSI Standard X9.42-2001, Public Key Cryptography for the Financial Services Industry: Agreement of Symmetric Keys Using Discrete Logarithm Cryptography for generation of the domain parameters[7];

b) Case: RSA Digital Signature Algorithm (with odd e)

ANSI X 9.31-1998 (May 1998), Digital Signatures Using Reversible Public Key Cryptography For The Financial Services Industry (rDSA)[8];

c) Case: Elliptic Curve Digital Signature Algorithm

ANSI X9.62-1998 (10 Oct 1999), Public Key Cryptography for the Financial Services Industry: Elliptic Curve Digital Signature Algorithm (ECDSA)

Application Note: A key size of 2048 bits is acceptable.

Application Note: For elliptic curve-based schemes the key size refers to the $\log_2$ of the order of the base point. As the preferred approach for cryptographic signature, elliptic curves will be required within a TBD time frame after all the necessary standards and other supporting information are fully established.

ADO_DEL_EXP.2.5D The developer shall use cryptographic hashing functions that employ a NIST-approved hash implementation of the Secure Hash algorithm and message digest size of at least 256 bits that meets the following: FIPS PUB 180-2.

ADO_DEL_EXP.2.1C The delivery documentation shall describe all procedures that are necessary to maintain security when distributing versions of the TOE to a user's site.

ADO_DEL_EXP.2.2C The delivery documentation shall describe how the various procedures and mechanisms provide for the detection of modifications, or any discrepancy between the developer's master copy and the version received at the user site.

ADO_DEL_EXP.2.3C The delivery documentation shall describe how the various procedures allow detection of attempts to masquerade as the developer, even in cases in which the developer has sent nothing to the user's site.

[6] FIPS PUB 186-3 is under development. It will incorporate the signature creation and verification processing of FIPS PUB 186-2, and the generation of domain parameters of ANSI X9.42. FIPS PUB 186-3 shall be used here when it is finalized and approved.

[7] Any pseudorandom RNG used in these schemes for generating private values shall be seeded by a nondeterministic RNG (both types of RNGs meeting RNG requirements in this PP).

[8] See previous footnote.

ADO_DEL_EXP.2.4C The delivery documentation shall describe how independent delivery channels are used to deliver the TOE code and to deliver the cryptographic keying materials used to verify the delivery of the code.

ADO_DEL_EXP.2.5C The delivery documentation shall describe how to use cryptographic mechanisms used to detect modification of the code of the TOE during the initial delivery and subsequent updates.

ADO_DEL_EXP.2.6C The delivery documentation shall describe how to use cryptographic mechanisms used to verify the integrity of the code of the TOE to ensure that the on-site version matches the master distribution version.

Application Note: It is assumed that the "cryptographic seal" of the TOE code will be verified when the TOE code is received from the TOE developer and protected appropriately at the user's site prior to loading into non-volatile memory for inclusion into the hosting hardware. However, for IT environments that cannot guarantee physical protection, additional procedures to re-check the integrity of the TOE code prior to loading should be provided by the IT environment.

ADO_DEL_EXP.2.7C The delivery documentation shall describe how to use the cryptographic mechanisms that interact with the TOE to verify a guarantee delivery from the intended source.

ADO_DEL_EXP.2.1E The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

ADO_DEL_EXP.2.2E The evaluator shall determine that the procedures provided result in a trusted delivery.

6.2.2 Installation, Generation and Start-Up (ADO_IGS)

6.2.2.1 Installation, Generation and Start-Up Procedures (ADO_IGS.1)

Application Note: This section is intended to address the requirements for configuring the TOE to be in a TOE Evaluated Configuration (TEC). Requirements for administrator guidance to correctly use TOE mechanisms (e.g., boot, initialization) to achieve an initial secure state are addressed in AGD_ADM.

ADO_IGS.1.1D The developer shall document procedures necessary for the secure installation, generation, and start-up of the TOE.

ADO_IGS.1.1C The documentation shall describe the steps necessary for secure installation, generation, and start-up of the TOE.

ADO_IGS.1.1E The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

ADO_IGS.1.2E The evaluator shall determine that the installation, generation, and start-up procedures result in a secure configuration.

6.3 Development (ADV)

6.3.1 Architectural Design (ADV_ARC)

6.3.1.1 Explicit: Architectural Design (ADV_ARC_EXP.1)

ADV_ARC_EXP.1.1D The developer shall provide the architectural design of the TSF.

ADV_ARC_EXP.1.1C The presentation of the architectural design of the TSF shall be informal.

ADV_ARC_EXP.1.2C The architectural design shall be internally consistent.

ADV_ARC_EXP.1.3C The architectural design shall describe the design of the TSF self-protection mechanisms.

ADV_ARC_EXP.1.4C The architectural design shall describe the design of the TSF in detail sufficient to determine that the security enforcing mechanisms cannot be bypassed.

ADV_ARC_EXP.1.5C The architectural design shall justify that the design of the TSF achieves the self-protection function.

ADV_ARC_EXP.1.6C The architectural design shall justify that the design of the TSF achieves the principle of least privilege specified in ADV_INT.

ADV_ARC_EXP.1.1E The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

ADV_ARC_EXP.1.2E The evaluator shall analyze the architectural design and other available TSF evidence to determine that FPT_SEP and FPT_RVM are accurately implemented in the TSF.

6.3.2 Composition Information (ADV_CMP)

6.3.2.1 Explicit: Detailed Composition Information (ADV_CMP_EXP.2)

ADV_CMP_EXP.2.1D The developer shall provide composition information addressed to system integrators.

ADV_CMP_EXP.2.1C The composition information shall describe the name, purpose, parameters, parameter definitions, and manner of use of all IT environment interfaces provided for use by the TSF.

ADV_CMP_EXP.2.2C The composition information shall describe how each TSFI can be invoked to use the IT environment interfaces.

ADV_CMP_EXP.2.1E The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

6.3.3 Functional Specification (ADV_FSP)

6.3.3.1 Explicit: Formal Functional Specification With Indirect Error Mapping (ADV_FSP_EXP.6)

ADV_FSP_EXP.6.1D The developer shall provide a functional specification.

ADV_FSP_EXP.6.1C The functional specification shall completely represent the TSF.

ADV_FSP_EXP.6.2C The functional specification shall be internally consistent.

ADV_FSP_EXP.6.3C The functional specification shall describe the external TSF interfaces (TSFI) using a formal style, supported by informal, explanatory text where appropriate.

ADV_FSP_EXP.6.4C The functional specification shall designate each external TSFI as security enforcing or security supporting.

ADV_FSP_EXP.6.5C The functional specification shall describe the purpose and method of use of each external TSFI.

ADV_FSP_EXP.6.6C The functional specification shall identify and describe all parameters associated with each external TSFI.

ADV_FSP_EXP.6.7C The functional specification shall describe all effects and all exceptions associated with each external TSFI.

ADV_FSP_EXP.6.8C The functional specification shall describe all error messages resulting from the effects and exceptions associated with each external TSFI.

ADV_FSP_EXP.6.9C The functional specification shall indicate the TSFI associated with each indirect error message.

ADV_FSP_EXP.6.1E The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

ADV_FSP_EXP.6.2E The evaluator shall determine that the functional specification is an accurate and complete instantiation of the user-visible TOE security functional requirements.

6.3.4 Development Requirements for Hardware

6.3.4.1 TBD

6.3.5 High-Level Design (ADV_HLD)

6.3.5.1 Explicit: Semiformal High-Level Design (ADV_HLD_EXP.4)

ADV_HLD_EXP.4.1D The developer shall provide the high-level design of the TOE

ADV_HLD_EXP.4.1C The high-level design shall describe the structure of the TOE in terms of subsystems.

ADV_HLD_EXP.4.2C The high-level design shall be internally consistent.

ADV_HLD_EXP.4.3C The presentation of the high-level design of the TSF shall be in semiformal style, supported by informal, explanatory text where appropriate.

ADV_HLD_EXP.4.4C The high-level design shall describe the design of the TOE in sufficient detail to determine what subsystems of the TOE are parts of the TSF.

ADV_HLD_EXP.4.5C The high-level design shall provide a description of the security functionality to be provided by the IT Environment.

ADV_HLD_EXP.4.6C The high-level design shall identify all subsystems in the TSF, and designate them as either security enforcing or security supporting.

ADV_HLD_EXP.4.7C The high-level design shall describe the structure of all subsystems of the TSF.

ADV_HLD_EXP.4.8C The high-level design shall describe the design of all behavior for all subsystems of the TSF.

ADV_HLD_EXP.4.9C The high-level design shall describe any interactions between the subsystems of the TSF.

ADV_HLD_EXP.4.1E The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

ADV_HLD_EXP.4.2E The evaluator shall determine that the high-level design is an accurate and complete instantiation of all user-visible TOE security functional requirements, with the exception of FPT_SEP and FPT_RVM.

6.3.6 Information Availability (ADV_IFA)

6.3.6.1 Explicit: Availability of Interface Information (ADV_IFA_EXP.1)

ADV_IFA_EXP.1.1D The developer shall provide the composition information to system integrators.

ADV_IFA_EXP.1.2D The developer shall provide the TSF functional specification to system integrators.

ADV_IFA_EXP.1.3D The developer shall provide the TSF test coverage analysis and TSF test procedure descriptions to the system integrators.

ADV_IFA_EXP.1.4D The developer shall provide an Integrators' Disclosure Agreement.

ADV_IFA_EXP.1.1C The Integrators' Disclosure Agreement shall identify the documentation comprising the composition information, TSF functional specification, TSF test coverage analysis, and TSF test procedure descriptions to be supplied under the terms of the agreement.

ADV_IFA_EXP.1.2C The Integrators' Disclosure Agreement shall specify terms under which the TSF functional specification, TSF test coverage analysis, and TSF test procedure descriptions will be made available to system integrators.

ADV_IFA_EXP.1.3C The terms specified in the Integrators' Disclosure Agreement shall not place onerous requirements on system integrators in order to obtain the TSF functional specification and TSF test coverage analysis.

ADV_IFA_EXP.1.1E The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

ADV_IFA_EXP.1.2E The evaluator shall confirm that the documentation comprising the composition information, TSF functional specification, TSF test coverage analysis, and TSF test procedure descriptions identified in the Integrators' Disclosure Agreement completely and accurately reflects the evidence supplied by the developer during the evaluation of the TOE.

6.3.7 Implementation Representation (ADV_IMP)

6.3.7.1 Explicit: Verified Implementation of the TSF (ADV_IMP_EXP.3)

ADV_IMP_EXP.3.1D The developer shall provide the implementation representation for the entire TSF.

ADV_IMP_EXP.3.2D The developer shall provide any additional information necessary to interpret the implementation representation supplied.

ADV_IMP_EXP.3.3D The developer shall supply the implementation of the software and firmware portions of the TSF.

ADV_IMP_EXP.3.4D The developer shall provide implementation information for the software and firmware portions of the TSF implementation.

ADV_IMP_EXP.3.5D The developer shall supply tools, and their associated documentation, used to debug the software and firmware portions of the TSF implementation.

ADV_IMP_EXP.3.1C The implementation representation shall unambiguously define the TSF to a level of detail such that the TSF can be generated without further design decisions, and are suitable so that they could be directly transformed to the implementation itself.

ADV_IMP_EXP.3.2C The additional information shall describe any conventions, directives, or other constructs necessary to determine the portions of the implementation representation that will be used when the implementation representation is transformed into the implementation itself.

ADV_IMP_EXP.3.3C The implementation information shall describe the format of the external representation of the implementation.

ADV_IMP_EXP.3.4C The implementation information shall map the implementation representation to the external representation of the implementation.

ADV_IMP_EXP.3.5C The implementation information shall provide a detailed description of the process(es) by which the external representation of the implementation is loaded and executed.

ADV_IMP_EXP.3.6C The documentation of the tools used for debugging shall be sufficient to allow use of the debugging tools for investigating the behavior of the TSF.

ADV_IMP_EXP.3.1E The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

ADV_IMP_EXP.3.2E The evaluator shall determine that the implementation representation is an accurate and complete instantiation of the TOE security functional requirements.

ADV_IMP_EXP.3.3E The evaluator shall confirm, through use of the supplied debugging tools and direct examination of the implementation, that the implementation provided is an accurate and complete instantiation of the TOE security functional requirements.

6.3.8 Trusted Initialization (ADV_INI)

6.3.8.1 Explicit: Trusted Initialization (ADV_INI_EXP.1

ADV_INI_EXP.1.1 The TOE shall provide procedures and/or mechanisms to ensure that the TSF configuration data reflects the user’s intention regarding partitioning and information flow.

ADV_INI_EXP.1.2 The TOE shall provide cryptographic mechanisms using TSF-provided cryptographic functions to detect modification of the TSF configuration data after it is created.

ADV_INI_EXP.1.3D The TOE shall provide cryptographic mechanisms using TSF-provided cryptographic functions to verify the integrity of the TSF configuration data prior to loading into non-volatile memory for inclusion into the hosting hardware.

ADV_INI_EXP.1.4D The TOE shall provide cryptographic mechanisms using TSF-provided cryptographic functions to detect modification of the code of the TOE during the initial delivery and subsequent updates.

ADV_INI_EXP.1.5D The TOE shall provide cryptographic mechanisms using TSF-provided cryptographic functions to verify the integrity of the code of the TOE to ensure that the on-site version matches the master distribution version.

ADV_INI_EXP.1.6D The TOE shall provide cryptographic mechanisms to guarantee delivery from the intended source.

ADV_INI_EXP.1.7D The TOE shall provide a boot mechanism to bring the TSF implementation (code) and TSF data into the TSF security domain, and an initialization mechanism to initialize the TSF to an initial secure state.

Application Note: See Boot and Initialization functions in Figure 2-2.

ADV_INI_EXP.1.8D The TOE shall provide an initialization mechanism to provide restrictive defaults for all security attributes that are not explicitly set the administrator.

ADV_INI_EXP.1.9D The TOE shall provide an initialization mechanism that shall [assignment: *list of the actions to take*] upon detection of the following conditions: [assignment: *list of detectable initialization errors and failures that prevent the establishment of an initial secure state for the TSF.*]

Application Note: This requirement is intended to provide assurance that the initialization code is capable of detecting and handling anomalies during the boot process during which the initial secure state is not yet established.

ADV_INI_EXP.1.10D The TOE shall provide a load mechanism to correctly convert to the TSF code and/or configuration data into a TSF-usable form.

ADV_INI_EXP.1.1E The evaluator shall determine that the configuration data generation mechanisms provided result in a correct configuration data.

ADV_INI_EXP.1.2E The evaluator shall determine that the delivery mechanisms provided result in a trusted delivery.

ADV_INI_EXP.1.3E The evaluator shall determine that the delivery mechanisms provided result in a trusted delivery.

ADV_INI_EXP.1.4E The evaluator shall determine that the boot and initialization mechanisms provided result in an initial secure state of the TOE.

ADV_INI_EXP.1.5E The evaluator shall determine that the load mechanism provided result in a correct conversion of the TSF code and/or configuration data into a TSF-useable form.

6.3.9 TSF Internals (ADV_INT)

6.3.9.1 Explicit: Minimization of Complexity (ADV_INT_EXP.4)

ADV_INT_EXP.4.1D The developer shall design and structure the TSF using modular decomposition.

ADV_INT_EXP.4.2D The developer shall use sound software engineering principles to achieve the modular decomposition of the TSF.

ADV_INT_EXP.4.3D The developer shall design the modules such that they exhibit good internal structure and are not overly complex.

ADV_INT_EXP.4.4D The developer shall design all TSF modules such that they exhibit only functional, sequential, or communicational cohesion, with limited exceptions.

ADV_INT_EXP.4.5D The developer shall design all TSF modules such that they exhibit only call coupling, with limited exceptions of common coupling.

ADV_INT_EXP.4.6D The developer shall implement TSF modules using coding standards that result in good internal structure that is not overly complex.

ADV_INT_EXP.4.7D The developer shall design and structure the TSF in a layered fashion that minimizes mutual dependencies between the layers of the design.

ADV_INT_EXP.4.8D The developer shall design and structure the TSF such that interactions between layers are initiated from a higher layer in the hierarchy down to a lower layer in the hierarchy with limited exceptions.

ADV_INT_EXP.4.9D The developer shall ensure that unused or redundant code are excluded from the TSF, with limited exceptions.

ADV_INT_EXP.4.10D The developer shall ensure that code that is not relevant for enforcing or supporting the TSP(s) are excluded from the TSF modules, with limited exceptions.

ADV_INT_EXP.4.11D The developer shall provide a software architectural description.

ADV_INT_EXP.3.7D The developer shall design and structure the TSF in such a way that the principle of least privilege is achieved with respect to TSF modules.

ADV_INT_EXP.4.1C The software architectural description shall identify all the modules of the TSF.

ADV_INT_EXP.4.2C The TSF modules shall be identical to those described by the low level design (ADV_LLD.1.4C).

ADV_INT_EXP.4.3C The software architectural description shall describe the process used for modular decomposition.

ADV_INT_EXP.4.4C The software architectural description shall describe how the TSF design is a reflection of the modular decomposition process.

ADV_INT_EXP.4.5C The software architectural description shall include the coding standards used in the development of the TSF.

ADV_INT_EXP.4.6C The software architectural description shall provide a justification, on a per module basis, of any deviations from the coding standards.

ADV_INT_EXP.4.7C The software architectural description shall include a coupling analysis that describes intermodule coupling for all TSF modules.

ADV_INT_EXP.4.8C The software architectural description shall provide a justification, on a per module basis, for any coupling or cohesion exhibited by modules of the TSF, other than those permitted.

ADV_INT_EXP.4.9C The software architectural description shall provide a justification, on a per module basis, that the modules of the TSF are not overly complex.

ADV_INT_EXP.4.10C The software architectural description shall describe the layering architecture and shall describe the services that each layer provides.

ADV_INT_EXP.4.11C The software architectural description shall identify the modules contained in each layer.

ADV_INT_EXP.4.12C The software architectural description shall identify all interactions between layers of the TSF.

ADV_INT_EXP.4.13C The software architectural description shall provide a justification of interactions that are initiated from a lower layer to a higher layer.

ADV_INT_EXP.4.14C The software architectural description shall show that mutual dependencies have been minimized, and justify those that remain.

ADV_INT_EXP.4.15C The software architectural description shall describe how the entire TSF has been structured to minimize complexity.

ADV_INT_EXP.4.16C The software architectural description shall justify the inclusion of any unused or redundant code in the TSF.

ADV_INT_EXP.4.16C The software architectural description shall justify the inclusion of any non-TSP-enforcing modules in the TSF.

ADV_INT_EXP.4.17C The software architectural description shall describe how the entire TSF has be structured to achieve least privilege.

ADV_INT_EXP.4.1E The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

ADV_INT_EXP.4.2E The evaluator shall perform a cohesion analysis for the modules that substantiates the type of cohesion claimed for a subset of TSF modules.

ADV_INT_EXP.4.3E The evaluator shall perform a complexity analysis for all TSF modules.

ADV_INT_EXP.3.4E The evaluator shall confirm that the entire TSF has been structured to achieve least privilege.

6.3.10 Low-level Design (ADV_LLD)

6.3.10.1 Explicit: Semi-Formal Low-Level Design (ADV_LLD_EXP.4)

ADV_LLD_EXP.4.1D The developer shall provide the low-level design of the TSF.

ADV_LLD_EXP.4.1C The presentation of the low-level design shall be semi-formal style, supported by informal, explanatory text where appropriate.

ADV_LLD_EXP.4.2C The presentation of the low-level design shall be separate from the implementation representation.

ADV_LLD_EXP.4.3C The low-level design shall be internally consistent.

ADV_LLD_EXP.4.4C The low-level design shall identify and describe data that are common to more than one module, where any of the modules is a security-enforcing module.

ADV_LLD_EXP.4.5C The low-level design shall describe the TSF in terms of modules.

ADV_LLD_EXP.4.6C The low-level design shall describe each module in terms of its purpose, interfaces, return values from those interfaces, called interfaces to other modules, and global variables.

ADV_LLD_EXP.4.7C For each module, the low-level design shall provide an algorithmic description detailed enough to represent the TSF implementation.

Application Note: An algorithmic description contains sufficient detail such that two different programmers would produce functionally-equivalent code, although data structures, programming methods, etc. may differ.

ADV_LLD_EXP.4.1E The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

ADV_LLD_EXP.4.2E The evaluator shall determine that the low-level design is an accurate and complete instantiation of all TOE security functional requirements, with the exception of FTP_SEP and FPT_RVM.

6.3.11 Representation Correspondence (ADV_RCR)

6.3.11.1 Explicit: Formal Correspondence Demonstration (ADV_RCR_EXP.3)

Application Note: The developer must either demonstrate or prove correspondence, as described in the requirements below, commensurate with the level or rigor of presentation style. For example, correspondence must be proven when corresponding representations are formally specified.

ADV_RCR_EXP.3.1D The developer shall provide a correspondence between all the SFRs specified in the ST and the TSFIs specified in the functional specification.

ADV_RCR_EXP.3.2D The developer shall provide a correspondence between the TSFIs and all the subsystems defined in the high-level design.

ADV_RCR_EXP.3.3D The developer shall provide a correspondence between the subsystems and the modules defined in the low-level design.

ADV_RCR_EXP.3.4D The developer shall provide a correspondence between the modules and the implementation representation.

ADV_RCR_EXP.3.5D The developer shall provide a correspondence analysis that the SFRs are completely and accurately realized by the TSFI.

ADV_RCR_EXP.3.1C The correspondence shall indicate where the functionality presented at the more abstract TSF representation is reflected in the less abstract TSF representation.

ADV_RCR_EXP.3.2C The correspondence analysis between the SFRs and the TSFI shall be formal.

ADV_RCR_EXP.3.1E The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

ADV_RCR_EXP.3.2E The evaluator shall determine the accuracy of the proofs of correspondence by selectively verifying the formal analysis.

6.3.12 Security Policy Modeling (ADV_SPM)

6.3.12.1 Explicit: Formal TOE Security Policy Model (ADV_SPM_EXP.3)

ADV_SPM_EXP.3.1D The developer shall provide a TSP model.

ADV_SPM_EXP.3.2D The developer shall demonstrate correspondence between the functional specification and the following policies of the TSP model: [assignment: *list of semiformally-stated policies TSP model*].

ADV_SPM_EXP.3.3D The developer shall prove correspondence between the functional specification and the following policies of the TSP model: [assignment: *list of formally-stated policies TSP model*].

ADV_SPM_EXP.3.1C The TSP model shall formally describe the rules and characteristics of all policies of the TSP that can be formally modeled and shall semiformally describe the rules and characteristics of other policies of the TSP.

ADV_SPM_EXP.3.2C The TSP model shall include a rationale that demonstrates that it is consistent and complete with respect to all policies of the TSP.

ADV_SPM_EXP.3.3C The demonstration of correspondence between the TSP model and the functional specification shall show that all of the TSFI in the functional specification are consistent and complete with respect to the TSP model.

ADV_SPM_EXP.3.4C Where the functional specification is semiformal, the demonstration of correspondence between the TSP model and the functional specification shall be semiformal.

ADV_SPM_EXP.3.5C Where the functional specification is formal, the proof of correspondence between the semiformally-stated portions of the TSP model and the functional specification shall be semiformal.

ADV_SPM_EXP.3.6C Where the functional specification is formal, the proof of correspondence between the formally-stated portions of the TSP model and the functional specification shall be formal.

ADV_SPM_EXP.3.1E The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

6.4 Guidance Documents (AGD)

6.4.1 Administrator Guidance (AGD_ADM)

6.4.1.1 Explicit: Administrator Guidance (AGD_ADM_EXP.1)

AGD_ADM_EXP.1.1D The developer shall provide administrator guidance addressed to system administrative personnel.

AGD_ADM_EXP.1.1C The administrator guidance shall describe the administrative functions and interfaces available to the administrator of the TOE.

AGD_ADM_EXP.1.2C The administrator guidance shall describe how to administer the TOE in a secure manner.

AGD_ADM_EXP.1.3C The administrator guidance shall contain warnings about functions and privileges that should be controlled in a secure processing environment.

AGD_ADM_EXP.1.4C The administrator guidance shall describe all assumptions regarding user behavior that are relevant to secure operation of the TOE.

AGD_ADM_EXP.1.5C The administrator guidance shall describe all security parameters under the control of the administrator, indicating secure values as appropriate.

AGD_ADM_EXP.1.6C The administrator guidance shall describe each type of security-relevant event relative to the administrative functions that need to be performed, including changing the security characteristics of entities under the control of the TSF.

AGD_ADM_EXP.1.7C The administrator guidance shall be consistent with all other documentation supplied for evaluation.

AGD_ADM_EXP.1.8C The administrator guidance shall describe all security requirements for the IT environment that are relevant to the administrator.

AGD_ADM_EXP.1.9C The administrator guidance shall document procedures necessary for the correct generation of the TSF configuration data.

AGD_ADM_EXP.1.10C The administrator guidance shall describe the steps necessary for correct generation of the TSF configuration data.

AGD_ADM_EXP.1.11C The administrator guidance shall document procedures to grant to each subject in the TSC the most restrictive set of authorizations and information flows needed for the performance of authorized tasks.

AGD_ADM_EXP.1.12C The administrator guidance shall describe the steps necessary for granting to each subject in the TSC the most restrictive set of authorizations and information flows needed for the performance of authorized tasks.

AGD_ADM_EXP.1.13C The administrator guidance shall document procedures necessary for using the load mechanism to convert the TSF code and/or data into a TSF-usable form.

AGD_ADM_EXP.1.14C The administrator guidance shall describe the steps necessary for using the load mechanism to convert the TSF code and/or data into a TSF-usable form.

AGD_ADM_EXP.1.15C The administrator guidance shall document procedures necessary for using the boot and initialization mechanisms to bring the TSF into an initial secure state.

AGD_ADM_EXP.1.16C The administrator guidance shall describe the steps necessary for using the boot and initialization mechanisms to bring the TSF into an initial secure state.

AGD_ADM_EXP.1.1E The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

AGD_ADM_EXP.1.2E The evaluator shall determine that the configuration generation procedures result in TSF configuration data that reflects the user's intention regarding partitioning and information flow.

AGD_ADM_EXP.1.3E The evaluator shall determine that the TOE is configured to grant to each subject in the TSC the most restrictive set of authorizations and information flows needed for the performance of authorized tasks.

AGD_ADM_EXP.1.4E The evaluator shall determine that the load procedures result in a form of the TSF code and/or data that is useable by the TSF.

AGD_ADM_EXP.1.5E The evaluator shall determine that the boot and initialization procedures result in an initial secure state of the TSF.

6.4.2 User Guidance (AGD_USR)

6.4.2.1 User Guidance (AGD_USR.1)

AGD_USR.1.1D The developer shall provide user guidance.

AGD_USR.1.1C The user guidance shall describe the functions and interfaces available to the non-administrative users of the TOE.

AGD_USR.1.2C The user guidance shall describe the use of user-accessible security functions provided by the TOE.

AGD_USR.1.3C The user guidance shall contain warnings about user-accessible functions and privileges that should be controlled in a secure processing environment.

AGD_USR.1.4C The user guidance shall clearly present all user responsibilities necessary for secure operation of the TOE, including those related to assumptions regarding user behavior found in the statement of TOE security environment.

AGD_USR.1.5C The user guidance shall be consistent with all other documentation supplied for evaluation.

AGD_USR.1.6C The user guidance shall describe all security requirements for the IT environment that are relevant to the user.

AGD_USR.1.1E The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

6.5 Life Cycle Support (ALC)

6.5.1 Development Security (ALC_DVS)

6.5.1.1 Sufficiency of Security Measures (ALC_DVS.2)

ALC_DVS.2.1D The developer shall produce development security documentation.

ALC_DVS.2.1C The development security documentation shall describe all the physical, procedural, personnel, and other security measures that are necessary to protect the confidentiality and integrity of the TOE design and implementation in its development environment.

ALC_DVS.2.2C The development security documentation shall provide evidence that these security measures are followed during the development and maintenance of the TOE.

ALC_DVS.2.3C The evidence shall justify that the security measures provide the necessary level of protection to maintain the confidentiality and integrity of the TOE.

ALC_DVS.2.1E The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

ALC_DVS.2.2E The evaluator shall confirm that the security measures are being applied.

6.5.2 Flaw Remediation (ALC_FLR)

6.5.2.1 Systematic Flaw Remediation (ALC_FLR.3)

ALC_FLR.3.1D The developer shall document the flaw remediation procedures.

ALC_FLR.3.2D The developer shall establish a procedure for accepting, and acting upon user reports of security flaws and requests for corrections to those flaws.

ALC_FLR.3.3D The developer shall designate one or more specific points of contact for user reports and inquiries about security issues involving the TOE.

ALC_FLR.3.1C The flaw remediation procedures documentation shall describe the procedures used to track all reported security flaws in each release of the TOE.

ALC_FLR.3.2C The flaw remediation procedures shall require that a description of the nature and effect of each security flaw be provided, as well as the status of finding a correction to that flaw.

ALC_FLR.3.3C The flaw remediation procedures shall require that corrective actions be identified for each of the security flaws.

ALC_FLR.3.4C The flaw remediation procedures documentation shall describe the methods used to provide flaw information, corrections and guidance on corrective actions to TOE users.

ALC_FLR.3.5C The procedures for processing reported security flaws shall ensure that any reported flaws are corrected and the correction issued to TOE users.

ALC_FLR.3.6C The procedures for processing reported security flaws shall provide safeguards that any corrections to these security flaws do not introduce any new flaws.

ALC_FLR.3.7C The flaw remediation procedures shall include a procedure requiring timely responses for the automatic distribution of security flaw reports and the associated corrections to registered users who might be affected by the security flaw.

ALC_FLR.3.1E The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

6.5.3 Life Cycle Definition (ALC_LCD)

6.5.3.1 Standardized Life-Cycle Model (ALC_LCD.2)

ALC_LCD.2.1D The developer shall establish a life-cycle model to be used in the development and maintenance of the TOE.

ALC_LCD.2.2D The developer shall provide life-cycle definition documentation.

ALC_LCD.2.3D The developer shall use a standardized life-cycle model to develop and maintain the TOE.

ALC_LCD.2.1C The life-cycle definition documentation shall describe the model used to develop and maintain the TOE.

ALC_LCD.2.2C The life-cycle model shall provide for the necessary control over the development and maintenance of the TOE.

ALC_LCD.2.3C The life-cycle definition documentation shall explain why the model was chosen.

ALC_LCD.2.4C The life-cycle definition documentation shall explain how the model is used to develop and maintain the TOE.

ALC_LCD.2.5C The life-cycle definition documentation shall demonstrate compliance with the standardized life-cycle model.

ALC_LCD.2.1E The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

6.5.4 Tools and Techniques (ALC_TAT)

6.5.4.1 Compliance with Implementation Standards – All Parts (ALC_TAT.3)

ALC_TAT.3.1D The developer shall identify the development tools being used for the TOE.

ALC_TAT.3.2D The developer shall document the selected implementation-dependent options of the development tools.

ALC_TAT.3.3D The developer shall describe the implementation standards for all parts of the TOE.

ALC_TAT.3.1C All development tools used for implementation shall be well-defined.

Application Note: The development tools include the compiler used to generate the TOE.

ALC_TAT.3.2C The documentation of the development tools shall unambiguously define the meaning of all statements used in the implementation.

ALC_TAT.3.3C The documentation of the development tools shall unambiguously define the meaning of all implementation-dependent options.

Application Note: This documentation includes the compiler options used during the generation of the TOE.

ALC_TAT.3.1E The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

ALC_TAT.3.2E The evaluator shall confirm that the implementation standards have been applied.

6.6 Ratings Maintenance (AMA)

6.6.1 Assurance Maintenance Plan (AMA_AMP)

6.6.1.1 Explicit: Assurance Maintenance Plan (AMA_AMP_EXP.1)

AMA_AMP_EXP.1.1D - The developer shall provide an AM Plan.

AMA_AMP_EXP.1.1C - The AM Plan shall identify the assurance baseline.

AMA_AMP_EXP.1.2C - The AM Plan shall contain or reference a brief description of the TOE, including the security functionality it provides.

AMA_AMP_EXP.1.3C - The AM Plan shall characterize the types of changes to the assurance baseline that are covered by the plan.

AMA_AMP_EXP.1.4C - The AM Plan shall describe the planned TOM release-cycle.

AMA_AMP_EXP.1.5C - The AM Plan shall identify the planned schedule of AM audits and the conditions for the end of maintenance.

AMA_AMP_EXP.1.5C - The AM Plan shall justify the planned schedule of AM audits and the conditions for the end of maintenance.

AMA_AMP_EXP.1.6C - The AM Plan shall identify the processes that are necessary for assigning, and ensuring currency of knowledge of, individual(s) assuming the role of security analyst.

AMA_AMP_EXP.1.7C - The AM Plan shall define the relationship between the security analyst and the development of the evidence.

AMA_AMP_EXP.1.8C - The AM Plan shall identify the qualifications that are necessary for the individual(s) identified as the security analyst.

AMA_AMP_EXP.1.9C - The AM Plan shall describe the procedures by which changes to the assurance baseline will be identified.

AMA_AMP_EXP.1.10C - The AM Plan shall describe the procedures that are necessary to be applied to the TOM to maintain the assurance established for the certified TOE.

AMA_AMP_EXP.1.11C - The AM Plan shall describe the controls and mechanisms that are necessary to ensure that the procedures documented in the AM Plan are followed.

AMA_AMP_EXP.1.1E - The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

6.7 Testing (ATE)

6.7.1 Coverage (ATE_COV)

6.7.1.1 Rigorous Analysis of Coverage (ATE_COV.3)

ATE_COV.3.1D The developer shall provide an analysis of the test coverage.

ATE_COV.3.1C The analysis of the test coverage shall demonstrate the correspondence between the tests identified in the test documentation and the TSF as described in the functional specification.

ATE_COV.3.2C The analysis of the test coverage shall demonstrate that the correspondence between the TSF as described in the functional specification and the tests identified in the test documentation is complete.

ATE_COV.3.3C The analysis of the test coverage shall rigorously demonstrate that all external interfaces of the TSF identified in the functional specification have been completely tested.

ATE_COV.3.1E The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

6.7.2 Depth (ATE_DPT)

6.7.2.1 Testing: Low Level Design (ATE_DPT.2)

ATE_DPT.2.1D The developer shall provide the analysis of the depth of testing.

ATE_DPT.2.1C The depth analysis shall demonstrate that the tests identified in the test documentation are sufficient to demonstrate that the TSF operates in accordance with its high-level design and low-level design.

ATE_DPT.2.1E The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

6.7.3 Functional Tests (ATE_FUN)

6.7.3.1 Ordered Functional Testing (ATE_FUN.2)

ATE_FUN.2.1D The developer shall test the TSF and document the results.

ATE_FUN.2.2D The developer shall provide test documentation.

ATE_FUN.2.1C The test documentation shall consist of test plans, test procedure descriptions, expected test results and actual test results.

ATE_FUN.2.2C The test plans shall identify the security functions to be tested and describe the goal of the tests to be performed.

ATE_FUN.2.3C The test procedure descriptions shall identify the tests to be performed and describe the scenarios for testing each security function. These scenarios shall include any ordering dependencies on the results of other tests.

ATE_FUN.2.4C The expected test results shall show the anticipated outputs from a successful execution of the tests.

ATE_FUN.2.5C The test results from the developer execution of the tests shall demonstrate that each tested security function behaved as specified.

ATE_FUN.2.6C The test documentation shall include an analysis of the test procedure ordering dependencies.

ATE_FUN.2.1E The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

6.7.4 Independent Testing (ATE_IND)

6.7.4.1 Independent Testing – Complete (ATE_IND.3)

ATE_IND.3.1D The developer shall provide the TOE for testing.

ATE_IND.3.1C The TOE shall be suitable for testing.

ATE_IND.3.2C The developer shall provide an equivalent set of resources to those that were used in the developer's functional testing of the TSF.

ATE_IND.3.1E The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

ATE_IND.3.2E The evaluator shall test a subset of the TSF as appropriate to confirm that the TOE operates as specified.

ATE_IND.3.3E The evaluator shall execute all tests in the test documentation to verify the developer test results.

6.8 Vulnerability Assessment (AVA)

6.8.1 Covert Channel Analysis (AVA_CCA)

6.8.1.1 Explicit: Systematic Covert Channel Analysis (AVA_CCA_EXP.2)

AVA_CCA_EXP.2.1D The developer shall conduct a search for inter-partition covert channels for each partition flow control policy.

AVA_CCA_EXP.2.2D For the cryptographic module, the developer shall conduct a search for covert channels for the leakage of critical cryptographic security parameters whose disclosure would compromise the security provided by the module.

AVA_CCA_EXP.2.3D The developer shall provide covert channel analysis documentation.

AVA_CCA_EXP.2.1C The analysis documentation shall identify covert channels and estimate their capacity.

AVA_CCA_EXP.2.2C The analysis documentation shall describe the procedures used for determining the existence of covert channels, and the information needed to carry out the covert channel analysis.

AVA_CCA_EXP.2.3C The analysis documentation shall describe all assumptions made during the covert channel analysis.

AVA_CCA_EXP.2.4C The analysis documentation shall describe the method used for estimating channel capacity, based on worst case scenarios.

AVA_CCA_EXP.2.5C The analysis documentation shall describe the worst case exploitation scenario for each identified covert channel.

AVA_CCA_EXP.2.6C The analysis documentation shall provide evidence that the method used to identify covert channels is systematic.

AVA_CCA_EXP.2.1E The NSA evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

AVA_CCA_EXP.2.2E The NSA evaluator shall confirm that the results of the covert channel analysis show that the TOE meets its functional requirements.

AVA_CCA_EXP.2.3E The NSA evaluator shall selectively validate the covert channel analysis through testing.

6.8.2 Misuse (AVA_MSU)

6.8.2.1 Analysis and Testing for Insecure States (AVA_MSU.3)

AVA_MSU.3.1D The developer shall provide guidance documentation.

AVA_MSU.3.2D The developer shall document an analysis of the guidance documentation.

AVA_MSU.3.1C The guidance documentation shall identify all possible modes of operation of the TOE (including operation following failure or operational error), their consequences and implications for maintaining secure operation.

AVA_MSU.3.2C The guidance documentation shall be complete, clear, consistent and reasonable.

AVA_MSU.3.3C The guidance documentation shall list all assumptions about the intended environment.

AVA_MSU.3.4C The guidance documentation shall list all requirements for external security measures (including external procedural, physical and personnel controls).

AVA_MSU.3.5C The analysis documentation shall demonstrate that the guidance documentation is complete.

AVA_MSU.3.1E The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

AVA_MSU.3.2E The evaluator shall repeat all configuration and installation procedures, and other procedures selectively, to confirm that the TOE can be configured and used securely using only the supplied guidance documentation.

AVA_MSU.3.3E The evaluator shall confirm that the use of the guidance documentation allows all insecure states to be detected.

AVA_MSU.3.4E The evaluator shall confirm that the analysis documentation shows that guidance is provided for secure operations in all modes of operation of the TOE.

AVA_MSU.3.5E The evaluator shall perform independent testing to determine that an administrator or user, with an understanding of the guidance documentation, would reasonably be able to determine if the TOE is configured and operating in the manner that is insecure.

6.8.3 Strength of TOE Security Functions (AVA_SOF)

6.8.3.1 Strength of TOE Security Function Evaluation (AVA_SOF.1)

Application Note: The security functions, for which strength of function claims are made, are identified in section 5.2 (Cryptographic Support).

AVA_SOF.1.1D The developer shall perform a strength of TOE security function analysis for each mechanism identified in the ST as having a strength of TOE security function claim.

AVA_SOF.1.1C For each mechanism with a strength of TOE security function claim the strength of TOE security function analysis shall show that it meets or exceeds the minimum strength level defined in the PP/ST.

AVA_SOF.1.2C For each mechanism with a specific strength of TOE security function claim the strength of TOE security function analysis shall show that it meets or exceeds the specific strength of function metric defined in the PP/ST.

AVA_SOF.1.1E The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

AVA_SOF.1.2E The evaluator shall confirm that the strength claims are correct.

6.8.4 Vulnerability Analysis (AVA_VLA)

6.8.4.1 Highly Resistant (AVA_VLA.4)

AVA_VLA.4.1D The developer shall perform and document an analysis of the TOE deliverables searching for ways in which a user can violate the TSP.

AVA_VLA.4.2D The developer shall document the disposition of identified vulnerabilities.

AVA_VLA.4.1C The documentation shall show, for all identified vulnerabilities, that the vulnerability cannot be exploited in the intended environment for the TOE.

AVA_VLA.4.2C The documentation shall justify that the TOE, with the identified vulnerabilities, is resistant to obvious penetration attacks.

AVA_VLA.4.3C The evidence shall show that the search for vulnerabilities is systematic.

AVA_VLA.4.4C The analysis documentation shall provide a justification that the analysis completely addresses the TOE deliverables.

AVA_VLA.4.1E The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

AVA_VLA.4.2E The evaluator shall conduct penetration testing, building on the developer vulnerability analysis, to ensure the identified vulnerabilities have been addressed.

AVA_VLA.4.3E The evaluator shall perform an independent vulnerability analysis.

AVA_VLA.4.4E The evaluator shall perform independent penetration testing, based on the independent vulnerability analysis, to determine the exploitability of additional identified vulnerabilities in the intended environment.

AVA_VLA.4.5E The evaluator shall determine that the TOE is resistant to penetration attacks performed by an attacker possessing a high attack potential.

AVA_VLA.4.6E **Refinement**: **The NSA evaluator shall perform an independent vulnerability analysis and conduct independent penetration testing.** 1

End Notes

This section records the assurance requirements where deletions of Common Criteria text were performed.

1 An additional requirement was added in ADV_VLA.4. Rationale: ADV_VLA.4.6E was added to require two levels of independent vulnerability analysis and penetration testing that is necessary to support a high robustness evaluation.

AVA_VLA.4.6E **Refinement: The NSA evaluator shall perform an independent vulnerability analysis and conduct independent penetration testing**.

7. Rationale

104 This section provides the rationale for the selection, creation, and use of security objectives and requirements as defined in sections 4 and 5, respectively.

7.1 Security Objectives derived from Threats

105 Each of the identified threats to security is addressed by one or more security objectives. Table 7.1 below provides the mapping from security objectives to threats, as well as a rationale that discusses how the threat is addressed. Definitions are provided (in italics) below each threat and security objective so the PP reader can reference these without having to go back to sections 3 and 4.

Table 7.1 – Mapping of Security Objectives to Threats

Threat	Objectives Addressing Threat	Rationale
T.ADMIN_ERROR *An administrator may incorrectly install or configure the TOE, or install a corrupted TOE resulting in ineffective security mechanisms.*	O.ADMIN_GUIDANCE *The TOE will provide administrators with the necessary information for secure management of the TOE.* O.INSTALL_GUIDANCE *The TOE will be delivered with the appropriate installation guidance to establish and maintain TOE security.*	Improper or insufficient security policies and mechanisms might be implemented if the administrator is not properly trained. However, if the administrator is provided sufficient guidance for the installation [O.INSTALL_GUIDANCE], configuration, and management of the TOE [O.ADMIN_GUIDANCE], the threat that the administrator may incorrectly install, configure, or manage the TOE, in a way that undermines security, is reduced.
T.ALTERED_DELIVERY *The TOE may be corrupted or otherwise modified during delivery such that the on-site version does not match the master distribution version.*	O.TRUSTED_DELIVERY *The integrity of the TOE must be protected during the initial delivery and subsequent updates, and verified to ensure that the on-site version matches the master distribution version.* O.CRYPTOGRAPHIC_SERVICES *The TOE will use cryptographic mechanisms to protect the integrity of TOE code and data as it resides within the system and when it is transmitted to other systems. The TOE will also use cryptographic mechanisms to verify the integrity of the TSF code and configuration data during initialization. The cryptographic mechanism will use NIST FIPS validated cryptography as a baseline with additional NSA-approved methods.* O.CRYPTOGRAPHIC_PROTECTION *The TOE will support separation of the cryptography from the rest of the TSF.*	To mitigate this threat, O.TRUSTED_DELIVERY requires integrity protection of the TOE. Checking the integrity of the TOE during initial delivery and subsequent updates is sufficient to determine if the TOE is corrupted or modified. O.CRYPTOGRAPHIC_SERVICES requires the use of cryptographic integrity mechanisms to provide a greater level of confidence that the integrity of the code of the TOE is protected. O.CRYPTOGRAPHIC_PROTECTION affords additional protection for the cryptography. This additional protection helps to protect the cryptography against compromise from accidental interference (e.g. coding errors) and malicious untrusted subjects.

Threat	Objectives Addressing Threat	Rationale
T.BAD_RECOVERY *The TOE may be placed in an insecure state as a result of unsuccessful recovery from a system failure or discontinuity.*	O.RECOVERY *Procedures and/or mechanisms will be provided to assure that recovery, such as from system failure or discontinuity, is obtained without a protection compromise.* O.CRYPTOGRAPHIC_SERVICES *The TOE will use cryptographic mechanisms to protect the integrity of TOE code and data as it resides within the system and when it is transmitted to other systems. The TOE will also use cryptographic mechanisms to verify the integrity of the TSF code and configuration data during initialization. The cryptographic mechanism will use NIST FIPS validated cryptography as a baseline with additional NSA-approved methods.* O.CRYPTOGRAPHIC_PROTECTIO N *The TOE will support separation of the cryptography from the rest of the TSF.* O.CORRECT_TSF_OPERATION *The TOE will provide a capability to test the TSF to ensure the correct operation of the TSF during normal operation.* O.TSF_INTEGRITY *The TOE will be able to verify the integrity of the TSF code and data.*	If recovery from a system failure or discontinuity is unsuccessful, the security condition of the TOE may be unknown. To mitigate this threat, O.RECOVERY provides procedures and/or mechanisms to ensure that recovery without a protection compromise is obtained. O.CRYPTOGRAPHIC_SERVICES requires the use of cryptographic integrity mechanisms to provide a greater level of confidence that the integrity of the code and data of the TOE is protected. O.CRYPTOGRAPHIC_PROTECTION affords additional protection for the cryptography. This additional protection helps to protect the cryptography against compromise from accidental interference (e.g. coding errors) and malicious untrusted subjects. O.CORRECT_TSF_OPERATION requires tests to be performed during automated recovery to demonstrate the correct operation of the TSF's implementation and the integrity of the TSF (hardware and software). O.TSF_INTEGRITY provides the mechanisms to verify the integrity of the TSF code and data thus ensuring protection after a failure.
T.COVERT_CHANNEL _EXPLOIT *An unauthorized information flow may occur between partitions as a result of covert channel exploitation.*	O.COVERT_CHANNEL_ANALYSIS *The TOE will undergo appropriate covert channel analysis by NSA to demonstrate that the TOE meets its functional requirement.* OE.CHANNELS *If the residual risk from covert channels is a concern, the applications executing on the TOE must be trusted with assurance commensurate with the value of the IT assets protected by the TOE.*	Unauthorized information flow may occur between partitions as a result of covert channel exploitation. O.COVERT_CHANNEL_ANALYSIS mitigates this threat by validating the vendor's covert channel analysis through testing and analysis. OE.CHANNELS mitigates this threat by requiring that applications capable of exploiting residual channels are trusted not to do so.
T.CRYPTO _COMPROMISE *A malicious user or process may cause key, data or executable code associated with the cryptographic functionality to be inappropriately accessed (viewed, modified, or deleted),*	OE.PHYSICAL *Physical security will be provided for the TOE by the IT environment commensurate with the value of the IT assets protected by the TOE.* O.CRYPTOGRAPHIC_PROTECTION *The TOE will support separation of the cryptography from the rest of the TSF.*	The cryptography is afforded external protection from viewing, modification, or deletion by malicious users through physical security measures provided by the IT environment [OE.PHYSICAL]. Further, as part of the TOE's security functions (TSF), the cryptography is afforded internal protection from viewing, modification, or

Threat	Objectives Addressing Threat	Rationale
thus compromising the cryptographic mechanisms and the data protected by those mechanisms.	O.REFERENCE_MONITOR *The TOE will maintain a domain for its own execution that protects itself and its resources from external interference, tampering, or unauthorized disclosure.*	deletion by malicious processes and users through the domain isolation maintained by the TOE for its own execution [O.REFERENCE_MONITOR]. Within the TSF's domain an additional protection is applied to the cryptography [O.CRYPTOGRAPHIC_PROTECTION]. This additional protection helps to protect the cryptography against compromise from accidental interference (e.g. coding errors) and malicious untrusted subjects.
T.INCORRECT_BOOT *The TSF implementation and TSF data are not correctly transferred into the TSF's execution domain.*	O.CORRECT_BOOT *The TOE will provide mechanisms to correctly transfer the TSF implementation and TSF data into the TSF's execution domain.*	O.CORRECT_BOOT mitigates this threat by requiring the TOE to provide mechanisms to correctly transfer the TSF implementation and TSF data into the TSF's execution domain as part of initialization.
T.INCORRECT_CONFIG *The TSF configuration data does not accurately reflect the user's intentions regarding partitioning and information flow.*	O.CORRECT_CONFIG *The TOE will provide procedures and mechanisms to generate the TSF configuration data such that the TSF configuration data accurately reflect the user's intentions regarding partitioning and information flow.*	O.CORRECT_CONFIG mitigates this threat by requiring the mechanisms used to generate the TSF configuration data (e.g., a configuration data generation tool) be included as part of the TOE. Since the TSF's policy enforcement mechanisms depend on the correctness of the TSF configuration data, it is important that the mechanisms used to generate the configuration data are subjected to analysis and testing with developmental assurance commensurate with the rest of the TOE.
T.INCORRECT_LOAD *The TSF code and/or configuration data are not correctly converted into a TSF-useable form.*	O.CORRECT_LOAD *The TOE will provide procedures and mechanisms to correctly convert the TSF code and/or configuration data into a TSF-useable form.*	O.CORRECT_LOAD mitigates this threat by requiring the mechanisms used to convert the TSF code and/or configuration data into a TSF-useable form be included as part of the TOE. Although these mechanisms are used off-line, they must be developed with a level of assurance commensurate with the rest of the TOE, such that the integrity of the TSF code and data can be preserved.
T.INSECURE_STATE *When the TOE is initially started or restarted after a failure, the security state of the TOE may be in an insecure state.*	O.SECURE_STATE *The TOE will provide mechanisms to transition the TSF to a secure state during start-up, re-activation of the current flow policy configuration data and activation of a new flow policy configuration data.* O.TSF_INTEGRITY *The TOE will be able to verify the integrity of the TSF code and data.*	To mitigate this threat, O.SECURE_STATE requires the TOE to provide the mechanisms to initialize the TSF into a secure state during start-up, re-activation of the current flow policy configuration data and activation of a new flow policy configuration data. O.TSF_INTEGRITY provides the mechanisms to verify the integrity of the TSF code and data thus ensuring a secure state after a failure or upon start-up.
T.POOR_DESIGN	O.CHANGE_MANAGEMENT	Intentional or unintentional errors may occur in the requirement specification, design or

Threat	Objectives Addressing Threat	Rationale
Unintentional or intentional errors in requirements specification or design of the TOE may occur, leading to flaws that may be exploited by a malicious subject.	*The configuration of, and all changes to, the TOE and its development evidence will be analyzed, tracked, and controlled throughout the TOE's development.* O.SOUND_DESIGN *The TOE will be designed using sound design principles and techniques. The TOE design, design principles and design techniques will be adequately and accurately documented.* O.VULNERABILITY_ANALYSIS_TEST *The TOE will undergo independent vulnerability analysis and penetration testing by NSA to demonstrate the design and implementation of the TOE does not allow attackers with high attack potential to violate the TOE's security policies.*	development of the TOE. To address this threat, O.SOUND_DESIGN requires sound design principles and techniques that help prevent flaws in the TOE's design by eliminating errors in the logic. In addition, O.CHANGE_MANAGEMENT addresses this threat by requiring all changes to the TOE and its development evidence be analyzed, tracked and controlled throughout the development cycle. To verify that there are no intentional or unintentional errors introduced in the design, O.VULNERABILITY_ANALYSIS_TEST demonstrates that the design of the TOE is resistant to attacks that exercise these design flaws and development errors.
T.POOR_IMPLEMENTATION *Unintentional or intentional errors in implementation of the TOE design may occur, leading to flaws that may be exploited by a malicious subject.*	O.CHANGE_MANAGEMENT *The configuration of, and all changes to, the TOE and its development evidence will be analyzed, tracked, and controlled throughout the TOE's development.* O.FUNCTIONAL_TESTING *The TOE will undergo independent security functional testing that demonstrates the TSF satisfies the security functional requirements.* O.SOUND_IMPLEMENTATION *The implementation of the TOE will be an accurate instantiation of its design.* O.VULNERABILITY_ANALYSIS_TEST *The TOE will undergo independent vulnerability analysis and penetration testing by NSA to demonstrate the design and implementation of the TOE does not allow attackers with high attack potential to violate the TOE's security policies.*	Intentional or unintentional errors may occur when implementing the design of the TOE. To address this threat, O.SOUND_IMPLEMENTATION ensures that the implementation is an accurate representation of the design. To ensure that an accurate representation of the design is maintained, O.CHANGE_MANAGEMENT ensures that all changes to the TOE and its development evidence are analyzed, tracked and controlled throughout the development cycle. To ensure that errors have not been introduced, O.FUNCTIONAL_TESTING validates that the TSF satisfies the security functional requirements. To further demonstrate that vulnerabilities are not present, O.VULNERABILITY_ANALYSIS_TEST ensures correct implementation of the TOE.
T.POOR_TEST *Lack of or insufficient tests to demonstrate that all TOE security functions operate correctly may result in incorrect TOE behavior being undiscovered.*	O.CORRECT_TSF_OPERATION *The TOE will provide a capability to test the TSF to ensure the correct operation of the TSF during normal operation.* O.FUNCTIONAL_TESTING *The TOE will undergo independent security functional testing that demonstrates the TSF satisfies the security functional requirements.* O.VULNERABILITY_ANALYSIS_TES	Design analysis determines that a TOE's documented design satisfies its security functional requirements. In order to ensure the TOE's design is correctly realized in its implementation, the appropriate level of functional testing of the TOE's security mechanisms must be performed during the evaluation of the TOE. O.FUNCTIONAL_TESTING ensures that independent functional testing is performed to demonstrate the TSF satisfies the security

Threat	Objectives Addressing Threat	Rationale
	T *The TOE will undergo independent vulnerability analysis and penetration testing by NSA to demonstrate the design and implementation of the TOE does not allow attackers with high attack potential to violate the TOE's security policies.*	functional requirements and the TOE's security mechanisms operate as documented. While functional testing serves an important purpose, it does not ensure the TSFI cannot be used in unintended ways to circumvent the TOE's security policies. O.VULNERABILITY_ANALYSIS_TEST addresses this concern by requiring a vulnerability analysis and penetration testing be performed by NSA. This objective provides a measure of confidence that the TOE does not contain security flaws that may not be identified through functional testing. While these testing activities are a necessary activity for successful completion of an evaluation, this testing activity does not address the concern that the TOE continues to operate correctly and enforce its security policies during normal operation. Some level of testing must be available to authorized users to ensure the TOE's security mechanisms continue to operate correctly once the TOE is fielded. O.CORRECT_TSF_OPERATION ensures that once the TOE is installed at a customer's location, the capability exists that the integrity of the TSF (hardware and software) can be demonstrated, and thus provides end users the confidence that the TOE's security policies continue to be enforced.
T.RESIDUAL_DATA *A subject may gain unauthorized access to data through reallocation of TOE resources from one subject to another.*	O.RESIDUAL_INFORMATION *The TOE will ensure that any data contained in a protected resource is not released when the resource is reallocated.*	The sharing of hardware resources such as primary and secondary storage components between subjects introduces the potential for information flow in violation of the TOE security policy when hardware resources are deallocated from one subject and allocated to another. In order to prevent such unintended consequences, the TOE prevents the compromise of the TOE security policy through mechanisms that ensure that residual information cannot be accessed after the resource has been reallocated (O.RESIDUAL_INFORMATION). The intent here is to prevent the unauthorized flow of information that would violate the TOE security policy.
T.RESOURCE _EXHAUSTION *A malicious subject may block others from system resources (e.g., system memory,*	O.RESOURCE_SHARING *The TOE shall provide mechanisms that mitigate attempts to exhaust TOE resources (e.g., system memory, persistent storage, and*	The sharing of resources (e.g., system memory, and processing time) between subjects introduces the potential for a malicious subject to obstruct another subject from access to resources via a resource exhaustion attack.

Threat	Objectives Addressing Threat	Rationale
persistent storage, and processing time) via a resource exhaustion attack.	*processing time).*	O.RESOURCE_SHARING mitigates this threat by requiring the TOE to allocate system resources to partitions according to the configuration data. The configuration data provides the capability to allocate resources such that over-allocation cannot occur.
T.TSF_COMPROMISE *A malicious subject may cause TSF data or executable code to be inappropriately accessed (viewed, modified, or deleted).*	OE.PHYSICAL *Physical security will be provided for the TOE by the IT environment commensurate with the value of the IT assets protected by the TOE.* O.REFERENCE_MONITOR *The TOE will maintain a domain for its own execution that protects itself and its resources from external interference, tampering, or unauthorized disclosure.*	The tampering with or destruction of TSF hardware, software, or configuration data via physical means is addressed by the physical security controls present in the TOE environment [OE.PHYSICAL]. O.REFERENCE_MONITOR addresses the threat of tampering with or destruction of TSF hardware, software, or configuration data by other (non-physical) means. It ensures that the TSF maintains a security domain for its own execution that protects it from interference and tampering by untrusted subjects and enforces the separation between the security domains of subjects within the TSC.
T.UNAUTHORIZED _ACCESS *A subject may gain access to resources and services for which it is not authorized according to the TOE security policy.*	OE.PHYSICAL *Physical security will be provided for the TOE by the IT environment commensurate with the value of the IT assets protected by the TOE.* O.ACCESS *The TOE will ensure that subjects gain only authorized access to resources that it controls.* O.PROTECT *The TOE will provide mechanisms to protect services and exported resources.*	Unauthorized users may physically tamper with the TOE hardware to gain unauthorized access to TOE resources. To mitigate this threat, OE.PHYSICAL restricts the physical access only to authorized personnel. Within the computing environment, O.ACCESS only allows subjects to gain access to resources for which they are authorized. At the same time, O.PROTECT provides mechanisms to provide self-protection for services and integrity protection for exported TSF data.

7.2 Objectives derived from Security Policies

106 Each of the identified security policies is addressed by one or more security objectives. Table 7.2 below provides the mapping from security objectives to security policies, as well as a rationale that discusses how the policy is addressed. Definitions are provided (in italics) below each policy and security objective so the PP reader can reference these without having to go back to sections 3 and 4.

Table 7.2 – Mapping of Security Objectives to Security Policies

Security Policy	Objectives Addressing Policy	Rationale

Security Policy	Objectives Addressing Policy	Rationale
P.ACCOUNTABILITY *The TOE shall provide the capability to make available information regarding the occurrence of security relevant events.*	O.AUDIT_GENERATION *The TOE will provide the capability to detect and generate audit records for security relevant auditable events.*	O.AUDIT_GENERATION supports this policy by requiring the TOE to detect and generate audit records upon the occurrence of security relevant events.
P.CRYPTOGRAPHY *The TOE shall use NIST FIPS validated cryptography as a baseline with additional NSA-approved methods for key management (i.e., generation, access, distribution, destruction, validation and packaging, handling, and storage of keys) and for cryptographic operations (i.e., encryption, decryption, signature, hashing, key exchange, and random number generation services).*	O.CRYPTOGRAPHIC_SERVICES *The TOE will use cryptographic mechanisms to protect the integrity of TOE code and data as it resides within the system and when it is transmitted to other systems. The TOE will also use cryptographic mechanisms to verify the integrity of the TSF code and configuration data during initialization. The cryptographic mechanism will use NIST FIPS validated cryptography as a baseline with additional NSA-approved methods.*	The TOE does not provide cryptographic services to applications. Hence this policy only applies to cryptographic mechanisms used internally by the TOE to satisfy the trusted delivery and trusted recovery requirements. O.CRYPTOGRAPHIC_SERVICES requires the use of validated and approved cryptographic methods for key management and cryptographic operations.
P.INDEPENDENT_TESTING *The TOE must undergo independent testing.*	O.FUNCTIONAL_TESTING *The TOE will undergo independent security functional testing that demonstrates the TSF satisfies the security functional requirements.* O.VULNERABILITY_ANALYSIS_TEST *The TOE will undergo independent vulnerability analysis and penetration testing by NSA to demonstrate the design and implementation of the TOE does not allow attackers with high attack potential to violate the TOE's security policies.*	This policy requires the TOE to undergo independent testing to verify its reliability and security. O.FUNCTIONAL_TESTING demonstrates the TSF satisfies the appropriate security functional requirements. O.VULNERABILITY_ANALYSIS_TEST requires the TOE to undergo vulnerability analysis and penetration testing by NSA to demonstrate the design and implementation of the TOE does not allow attackers with high attack potential to violate the TOE's security policies.
P.LEAST_PRIVILEGE *The TOE shall be designed such that the principle of least privilege is applied to limit the damage that can result from accident, error or unauthorized use.*	O.INTERNAL_LEAST_PRIVILEGE *The entire TSF will be structured to achieve the principle of least privilege among TSF modules.*	O.INTERNAL_LEAST_PRIVILEGE requires that the TSF be structured such that the principle of least privilege is applied to the internal software architecture of the TSF. A structured design facilitates the analysis of the TSF to ensure that the security policies are enforced, thus limiting the damage that can result from accident, error or unauthorized use.
P.RATINGS_MAINTENANCE *A plan for procedures and processes to maintain the TOE's rating must be in place to maintain the TOE's rating once it is evaluated..*	O.RATINGS_MAINTENANCE *Procedures and processes to maintain the TOE's rating will be documented.*	This policy requires the TOE developer to provide a plan that documents the procedures and processes to maintain the evaluated rating that is ultimately awarded the TOE. O.RATINGS_MAINTENANCE satisfies this policy by requiring the TOE developer to provide the required rating maintenance plan.

Security Policy	Objectives Addressing Policy	Rationale
P.SELECT_POLICY *The TOE shall provide the capability to select and activate a complete set of new policy configuration data. The TOE shall ensure that the policy enforced upon the activation of the new data is consistent with the new con figuration data.*	O.MANAGE *The TOE will provide all the functions necessary to support the administrative users and authorized subjects in their management of the configuration data, and restrict these functions from use by unauthorized subjects.* O.TSF_INTEGRITY *The TOE will be able to verify the integrity of the TSF code and data.*	This policy requires the TOE to allow the policy configuration to be modified as a single unit, either offline or during runtime. It also requires the TOE to ensure that after activation of the new policy, there are no flows extant that are inconsistent with the new policy. O.MANAGE restricts the online access to the policy selection and activation functions to authorized subjects. O.TSF_INTEGRITY supports this policy by ensuring the integrity of the new policy configuration.
P.SYSTEM_INTEGRITY *The TOE shall provide the ability to periodically validate its correct operation and, with the help of administrators if necessary, it must be able to recover from any errors that are detected.*	O.CORRECT_TSF_OPERATION *The TOE will provide a capability to test the TSF to ensure the correct operation of the TSF during normal operation.* O.RECOVERY *Procedures and/or mechanisms will be provided to assure that recovery, such as from system failure or discontinuity, is obtained without a protection compromise.*	This policy requires the TOE to 1) periodically test itself to provide some measure of confidence that the TOE is operating in accordance with its security policies, and 2) provide a means for the TOE to recover from detectable errors in its operation. O.CORRECT_TSF_OPERATION supports this policy by requiring the TOE to provide a capability to test the TSF to demonstrate the correct operation of the TSF in its operational environment. O.RECOVERY satisfies this policy by requiring the TOE to provide procedures and/or mechanisms to ensure that the TOE can recover from detectable errors.
P.USER_GUIDANCE *The TOE shall provide documentation regarding the correct use of the TOE security features.*	O.USER_GUIDANCE *The TOE shall provide users with the necessary information for secure use of the TOE.*	This policy requires that the TOE documentation provide adequate information for the secure use and operation of the TOE. O.USER_GUIDANCE satisfies this policy by requiring that the necessary user information be provided.
P.VULNERABILITY _ANALYSIS_AND_TEST *The TOE must undergo independent vulnerability analysis and penetration testing by NSA to demonstrate that the TOE is resistant to an attacker possessing a high attack potential.*	O.VULNERABILITY_ANALYSIS_ TEST *The TOE will undergo independent vulnerability analysis and penetration testing by NSA to demonstrate the design and implementation of the TOE does not allow attackers with high attack potential to violate the TOE's security policies.*	O.VULNERABILITY_ANALYSIS_TEST satisfies this policy by ensuring that an independent vulnerability analysis is performed on the TOE and penetration testing based on that analysis is performed. Having an independent party perform the analysis helps ensure objectivity and eliminates preconceived notions of the TOE's design and implementation that may otherwise affect the thoroughness of the analysis. The level of analysis and testing requires that an attacker with a high attack potential cannot compromise the TOE's ability to enforce its security policies.

7.3 Objectives derived from Assumptions

107 Each of the identified security assumptions is addressed by one or more security objectives. Table 7.3 below provides the mapping from security objectives to security assumptions, as well as a rationale that discusses how the assumption is addressed. Definitions are provided (in italics) below each assumption and security objective so the PP reader can reference these without having to go back to sections 3 and 4.

Table 7.3 – Mapping of Security Objectives to Assumptions

Assumption	Objectives Addressing Assumption	Rationale
A.CHANNELS *If the residual risk from covert channels is a concern, it is assumed that the applications executing on the TOE are trusted with assurance commensurate with the value of the IT assets protected by the TOE.*	OE.CHANNELS *If the residual risk from covert channels is a concern, the applications executing on the TOE must be trusted with assurance commensurate with the value of the IT assets protected by the TOE.*	OE.CHANNELS addresses this assumption by requiring that applications capable of exploiting residual channels are trusted not to do so. These applications are required to be trusted with assurance commensurate with the value of the IT assets protected by the TOE.
A.LEAST_PRIVILEGE *It is assumed that the TOE will be configured such that the principle of least privilege is applied to limit the damage that can result from accident, error or unauthorized use.*	OE.LEAST_PRIVILEGE *The TOE must be configured such that the principle of least privilege is applied to limit the damage that can result form accident, error or unauthorized use.*	OE.LEAST_PRIVILEGE addresses this assumption by requiring the IT environment to provide procedures to configure subjects in accordance with the principle of least privilege such that damages that can result from accident, error or unauthorized use could be effectively contained.
A.PHYSICAL *It is assumed that the IT environment provides the TOE with appropriate physical security commensurate with the value of the IT assets protected by the TOE.*	OE.PHYSICAL *Physical security will be provided for the TOE by the IT environment commensurate with the value of the IT assets protected by the TOE.*	OE.PHYSICAL addresses this assumption by requiring the IT environment to provide physical security for the TOE that is commensurate with the value of the IT assets protected by the TOE.
A.TRUSTED_FLOWS *If a subject is allowed by the configuration data to cause information flow in violation of the partial ordering of information flows between partitions, it is assumed that the subject is trusted with assurance commensurate with the value of the IT assets in all partitions to which it has access.*	OE.TRUSTED_FLOWS *If a subject is allowed by the configuration data to cause information flow in violation of the partial ordering of information flows between partitions, that subject must be trusted with assurance commensurate with the value of the IT assets in all partitions to which it has access.*	OE.TRUSTED_FLOWS addresses this assumption by requiring that a subjects capable of causing information flow in violation of the partial ordering of information flows between partitions be trusted with assurance commensurate with the value of the IT assets in all partitions to which it has access. The "partial ordering" requirement addresses a significant characteristic of the class of systems represented by this protection profile. Partitions between which flows occur in violation of the partial ordering result in a logical equivalence class of information in those partitions, since all information can be shared between the partitions. In some cases, flows between partitions in violation of the

Assumption	Objectives Addressing Assumption	Rationale
		partial ordering are useful when constructing an application if it can be assured that only certain information is permitted to flow in violation of the partial ordering. If a subject has insufficient assurance, then it may be assumed to cause unintended flows between the partitions. While the subject-to-resource flow controls can be used to prevent inter-partition flows otherwise allowed by the partition-to-partition flow rules, it is generally intractable to determine which information in a partition will be (e.g., transitively) allowed to flow into another partition once the flow is allowed by a partition-to-partition flow rule and a subject-to-resource flow rule (e.g., to support a guard or downgrader application). Therefore, if such an inter-partition flow were allowed, we would require of the environment that the subject (e.g., application) have a level of trust that is adequate to protect the information in both the source and the destination partitions.
A.TRUSTED_INDIVIDUAL *If an individual is allowed to perform procedures upon which the security of the TOE may depend, it is assumed that the individual is trusted with assurance commensurate with the value of the IT assets.*	OE.TRUSTED_INDIVIDUAL *If an individual is allowed to perform procedures upon which the security of the TOE may depend, that individual must be trusted with assurance commensurate with the value of the IT assets.*	OE.TRUSTED_INDIVIDUAL addresses this assumption by requiring that any individual who is allowed to perform procedures that affect the security of the TOE be trusted with assurance commensurate with the value of the IT assets. This requirement is allocated to the IT environment because there are no Identification & Authentication requirements for the TOE.

7.4 Requirements Rationale

108 Each of the TOE security objectives identified in section 4 are addressed by one or more security requirements. Table 7.4 below provides the mapping from security requirements to security objectives, as well as a rationale that discusses how the security objective is met. Definitions are provided (in italics) below each security objective so the PP reader can reference these without having to go back to section 4.

Table 7.4 – Mapping of Security Requirements to Objectives

Objectives from Policies/Threats	Requirements Meeting Objectives	Rationale

Objectives from Policies/Threats	Requirements Meeting Objectives	Rationale
O.ACCESS *The TOE will ensure that subjects gain only authorized access to resources that it controls.*	FDP_IFC.2 FDP_IFF.1 FPT_RVM.1	This objective requires the TOE to manage resources that it controls such that subjects can only gain access to those resources that they are permitted to use. The combination of FDP_IFC.2, FDP_IFF.1 and FPT_RVM.1 satisfies this objective. FDP_IFC.2 requires the TSF to enforce two SFPs, Information Flow Control policy and Partition Flow Control policy, on all subjects and partitions, all exported resources and all operations that cause information to flow to and from all subjects and between partitions. FDP_IFF.1 specifies the policy rules to be enforced by the TSF and the security attributes used by the enforcement rules. The Information Flow Control policy rule requires the TSF to permit an information flow between a subject and a resource only if it is explicitly allowed by the configuration data. Similarly the Partition Flow Control policy rule requires the TSF to permit an information flow between partitions only if it is explicitly allowed by the configuration data. The Separation rule requires the TSF to deny all information flows unless the requested flow is explicitly allowed by the configuration data. FPT_RVM.1 ensures that the TSF makes policy decisions on all attempts to access the TOE resources. Without this non-bypassability requirement, the TSF could not be relied upon to completely enforce the security policies.
O.ADMIN_GUIDANCE *The TOE will provide administrators with the necessary information for secure management of the TOE.*	ADO_IGS.1 AGD_ADM_EXP.1	ADO_IGS.1 requires the developer to provide the procedures necessary to install and start-up an instance of the TOE that was evaluated, i.e., an evaluated configuration of the TOE. AGD_ADM_EXP.1 requires the developer to provide administrative guidance to configure and administer the TOE securely for the IT environment within which it is intended to operate. The necessary information for secure management of the TOE include instructions on proper use of the administrative functions, warnings about functions and privileges that should be controlled, assumptions regarding user behavior, correct settings of security parameters, and security requirements for the IT environment.
O.AUDIT_GENERATIO N *The TOE will provide the capability to detect and generate audit records for security relevant auditable events.*	FAU_ARP.1 FAU_GEN.1 FAU_SEL.1 FPT_STM.1	FAU_ARP.1 requires the TSF to take the actions upon the detection of failures of TSF self tests. By allowing the ST author to assign the list of actions based on the intended use of the TOE, this PP affords design flexibility to the ST authors. FAU_GEN.1 defines the set of events for which the TOE must be able to generate audit records. This requirement also defines the minimum amount of data to be provided for each auditable event. Additionally, this requirement places a requirement on the level of audit detail on any additional security functional requirements an ST author adds to this PP. FAU_SEL.1 requires the TSF to generate audit records for selective auditable events based on a set of audit selection

Objectives from Policies/Threats	Requirements Meeting Objectives	Rationale
		attributes. This provides the administrator with the flexibility in detecting only those events that are deemed necessary by site policy, thus reducing the amount of resources consumed by the audit mechanism. FPT_STM.1 requires the TSF to provide reliable time stamps for its own use.
O.CHANGE_MANAGEMENT *The configuration of, and all changes to, the TOE and its development evidence will be analyzed, tracked, and controlled throughout the TOE's development.*	ACM_AUT.2 ACM_CAP.5 ACM_SCP.3 ALC_DVS.2 ALC_FLR.3 ALC_LCD.2 ALC_TAT.3	This objective is satisfied by the following Configuration Management (CM) and Life Cycle (LC) requirements. ACM_AUT.2 requires the TOE developer to have a CM plan and use a CM system that provides an automated means to control changes made to all configuration items that comprise the TOE, and to support the generation of the TOE. This requirement also requires the developer to describe in the CM plan the automated tools used in the CM system and how those tools are used in the CM system. Thus, ACM_AUT.2 aids in understanding how the CM system enforces the control over changes made to the TOE. ACM_CAP.5 requires the developer to describe in the CM plan how changes to the TOE and its evaluation deliverables are managed by the CM system. The CM system is required to operate in accordance with the CM plan and provide the capability to control who on the development staff can make changes to the TOE and its developed evidence. Furthermore, the CM system is required to enforce separation of duties (e.g., developers cannot be part of the CM staff), clearly identify the configuration items that comprise the TSF, and support the audit of modifications to the TOE. In addition to the CM plan and CM system, the developer is also required to provide a list of uniquely identified configuration items that comprise the TOE, an acceptance plan and integration procedures. The configuration list is used by the CM system to control unauthorized modification, addition, or deletion of the TOE configuration items, and by the integration procedures to ensure that the TOE is generated correctly. The acceptance plan describes how modified or newly created configuration items are reviewed and accepted as part of the TOE. The developer is required to justify that the acceptance procedures provide for an adequate and appropriate review of all changes to the TOE. This requirement satisfies the "analyzed" aspect of this objective. ACM_SCP.3 is necessary to define what items must be under the control of the CM system. This requirement ensures that the TOE implementation representation, design documentation, test documentation (including the executable test suite), user and administrator guidance, CM documentation, security flaws, and development tools (and related information) are tracked by the CM system. ALC_DVS.2 requires the developer to describe the security measures used in the development environment to ensure the integrity and confidentiality of the TOE. Furthermore, the

Objectives from Policies/Threats	Requirements Meeting Objectives	Rationale
		developer must also provide evidence that that these security measures are followed by the development team, and justify that these measures provide the necessary level of protection. The physical, procedural, and personnel security measures the developer uses provides an added level of control over who and how changes are made to the TOE and its associated evidence. ALC_FLR.3 requires the developer to track and correct flaws in the TOE that have been discovered either through developer actions (e.g., developer testing) or by others. In addition to correcting discovered flaws, the flaw remediation process used by the developer must also ensure that new flaws are not created while fixing the discovered flaws. The developer is also required to support automatic distribution of secure flaw reports and to timely inform users who might be affected by the discovered flaws. ALC_LCD.2 requires the developer to use a standardized life-cycle model that describes the procedures, tools and techniques used in the development and maintenance of the TOE. Procedural aspects such as design methods, code or documentation reviews, how changes to the TOE are reviewed and accepted or rejected will add assurance for the TOE at the time of the initial evaluation and during its maintenance phases. The developer is required to explain why the particular life cycle model is chosen and how it is used, and to demonstrate that the life cycle documentation is compliance with the life cycle model. ALC_TAT.3 ensures that all the tools and techniques used during the development and maintenance of the TOE are well defined including the selected implementation-dependent options of the development tools. It also requires the developer to establish implementation standards for all parts of the TOE. This will mitigate the risk of using ill-defined, inconsistent or incorrect development tools and techniques.
O.CORRECT_BOOT *The TOE will provide mechanisms to correctly transfer the TSF implementation and TSF data into the TSF's execution domain.*	AGD_ADM_EXP.1 ADV requirements for Tools <TBD>	AGD_ADM_EXP.1 requires the developer to provide administrator guidance for the proper use of the boot mechanism. <Rationale for ADV requirements for Tools are TBD>
O.CORRECT_CONFIG *The TOE will provide procedures and mechanisms to generate the TSF configuration data such that the TSF configuration data accurately reflects the user's intentions regarding partitioning and information flow.*	AGD_ADM_EXP.1 ADV requirements for Tools <TBD>	AGD_ADM_EXP.1 requires the developer to provide administrator guidance for the correct generation of the configuration data. <Rationale for ADV requirements for Tools are TBD>
O.CORRECT_LOAD	AGD_ADM_EXP.1	AGD_ADM_EXP.1 requires the developer to provide

Objectives from Policies/Threats	Requirements Meeting Objectives	Rationale
The TOE will provide procedures and mechanisms to correctly convert the TSF code and/or configuration data into a TSF-useable form.	ADV requirements for Tools <TBD>	administrator guidance for the proper use of the load mechanism. <Rationale for ADV requirements for Tools are TBD>
O.CORRECT_TSF_OPERATION *The TOE will provide a capability to test the TSF to ensure the correct operation of the TSF during normal operation.*	FMT_MTD.3 FPT_AMT.1 FPT_TST_EXP.1	FMT_MTD.3 imposes requirements on the management of the TSF data other than security attributes. This requirement satisfies this objective by requiring the TSF to only accept values that fall within the defined range for the TSF data. FPT_AMT.1 provides the end user the ability to discover any failures in the hardware security mechanisms that could render the TSF ineffective in enforcing its security policies. This requirement requires the TSF to test the hardware security mechanism during the initial start-up and also periodically during normal operation. The standard FPT_TST.1 requirement only mandates the TSF to verify the integrity of the TSF data and TSF executable code stored in non-volatile storage. However for high robustness, it is necessary for the TSF to also verify the integrity of the TSF executable image loaded in memory. Hence, FPT_TST_EXP.1 was created. These integrity tests are necessary because the TSF may not correctly enforce its security policies if its data or code is corrupted. FPT_TST_EXP.1 also requires the TSF to run a suite of self-tests to verify the software portions of the TSF. This requirement explicitly specifies that the TSF self tests be run during the initial start-up, but leaves the conditions under which the self tests should occur during normal operation to be filled in by the ST author. This allows the ST author to tailor the testing requirements to be appropriate to conditions of the TSF's normal operation. The tests required by FPT_AMT.1 and FPT_TST_EXP.1 verify the correct operation and integrity of all three parts of the TSF, i.e., TSF's underlying abstract machine, TSF's implementation and TSF's data.
O.COVERT_CHANNEL_ANALYSIS *The TOE will undergo appropriate covert channel analysis by NSA to demonstrate that the TOE meets its functional requirement.*	AVA_CCA_EXP.2	AVA_CCA_EXP.2 requires the developer to perform a systematic search for inter-partition covert channels and potential cryptographic key leakage from the cryptographic module. It also requires the developer to document their analysis and provide the documentation as evaluation evidence. Since all subjects assigned to a partition are of the same equivalence class, a search for intra-partition covert channels is not needed. A thorough search for cryptographic key leakage is important because the TSF uses cryptography to protect itself as well as exported TSF data. A systematic search, as opposed to an informal search, is necessary because it is important that the covert channels be identified in a structured and repeatable way to aid the validation

Objectives from Policies/Threats	Requirements Meeting Objectives	Rationale
		of the covert channel analysis. AVA_CCA_EXP.2 also requires the NSA evaluator to confirm the results of the covert channel analysis and to selectively validate the covert channel analysis through testing. This will afford additional assurance evidence to support a high robustness evaluation.
O.CRYPTOGRAPHIC_PROTECTION *The TOE will support separation of the cryptography from the rest of the TSF.*	FPT_SEP.3	FPT_SEP.3 was refined to require the TSF to maintain a separate security domain for the information flow control, partition flow control and cryptography SFPs. Since cryptography is used by the TSF to protect the integrity of TSF data, including TSF flow policy data, the cryptography SFP should be given the same level of domain protection afforded to the information and partition flow control SFPs.
O.CRYPTOGRAPHIC_SERVICES *The TOE will use cryptographic mechanisms to protect the integrity of TOE code and data as it resides within the system and when it is transmitted to other systems.. The cryptographic mechanism will use NIST FIPS validated cryptography as a baseline with additional NSA-approved methods.*	FCS_BCM_EXP.1	Baseline cryptographic services are provided in the TOE by FIPS PUB 140-2 compliant modules implemented in hardware, in software, or in hardware/software combinations [FCS_BCM_EXP.1]. The cryptographic services offered by this baseline capability are augmented to support primarily digital signature and cryptographic hashing functions.
O.FUNCTIONAL_TESTING *The TOE will undergo independent security functional testing that demonstrates the TSF satisfies the security functional requirements.*	ATE_COV.3 ATE_DPT.2 ATE_FUN.2 ATE_IND.3	ATE_COV.3, ATE_DPT.2 and ATE_FUN.2 impose testing requirements on the developer to create and document the security test suite. ATE_IND.3 levies requirements on the evaluation team to independently verify the testing results. The combination of these requirements satisfies this objective. ATE_COV.3 requires the developer to provide an analysis of the test coverage to demonstrate that the TSF and TSF interfaces are completely addressed by the developer's test suite. While this requirement does not require exhaustive testing of the TSF, it does impose exhaustive testing of the TSF interfaces to ensure that the TSF interfaces meet the their security functional requirements. This component also requires an independent confirmation of the completeness of the test suite. ATE_DPT.2 requires the developer to provide an analysis of the depth of the functional testing to demonstrate that the TSF is implemented and operates as specified by its high-level design and low-level design. This component complements ATE_COV.2 by ensuring that the developer takes into account the high-level and low-level design when developing their test suite. ATE_FUN.2 requires the developer to test the TSF and provide document the results. The functional tests are required to be

Objectives from Policies/Threats	Requirements Meeting Objectives	Rationale
		loop-free. The developer's test documentation must include an analysis of the test procedure ordering dependencies to demonstrate the testing is not circular. The developer must provide sufficient test documentation, i.e., test plan, test procedures, and test results, to support independent verification of the test results and test coverage analysis. ATE_IND.3 requires the developer to provide the evaluator with the TOE and testing materials for independent testing. The developer must provide the same testing materials that were used by the developer to perform the developer's functional testing. These must include, minimally, test suite executables and source code, and machine-readable test documentation. ATE_IND.3 also levies testing requirements on the evaluator to verify the developer's test results by re-testing all tests performed by the developer, and to develop and run their own additional tests that exercise the TOE in areas that are not well demonstrated by the developer's test suite. By repeating all of the developer's tests and running their own test suite, the evaluator can demonstrate that the TSF satisfies all security functional requirements as required by this objective.
O.INSTALL_GUIDANCE *The TOE will be delivered with the appropriate installation guidance to establish and maintain TOE security.*	ADO_DEL_EXP.2 ADO_IGS.1	This objective is satisfied by the documentation requirements of the trusted delivery and secure installation and start-up functions. ADO_DEL_EXP.2 was created because none of the existing ADO_DEL components address the need to use cryptography to verify the delivery of the TOE code. ADO_DEL_EXP.2 was based on ADO_DEL.2 which requires the developer to describe the procedures and technical measures that the developer put in place to: 1) detect modifications during transit, 2) detect any discrepancy between the developer's master version and the delivered version, and 3) detect any attempts to masquerade as the developer. ADO_DEL_EXP.2 expanded the scope of ADO_DEL.2 by requiring the developer and to provide cryptographic mechanisms to protect the integrity of the TOE during delivery. ADO_DEL_EXP.2 also requires the developer to follow the developer-prescribed delivery procedures. After verifying that the TOE delivery from the developer is the right version and tamper-free, the user is responsible to configure and install the TOE in accordance with the TOE's intended use before running it. ADO_IGS.1 requires the developer to provide the guidance on how to use the installation and start-up procedures to install and start-up an instance of the TOE that was evaluated. ADO_IGS.1 further requires the evaluator to verify that if the procedures are used as described, they will result a secure installation and start-up of the TOE.
O.INTERNAL_LEAST_PRIVILEGE *The entire TSF will be*	ADV_INT_EXP.3	ADV_INT_EXP.3 was created because the existing ADV_INT.3 component does not address the need to apply the principle of least privilege (PoLP) to the design and structure of the TSF.

Objectives from Policies/Threats	Requirements Meeting Objectives	Rationale
structured to achieve the principle of least privilege among TSF modules.		ADV_INT_EXP.3 expanded the scope of ADV_INT.3 by requiring the developer to design and structure the TSF such that PoLP can be achieved. Together with layering and minimization, least privilege will afford simplicity to the implementation, which, in turn, provides a greater level of confidence in the analysis of the correctness of the implementation. ADV_INT_EXP.3 also require the evaluator to confirm that the TSF has been internally structured to achieve least privilege among TSF modules.
O.MANAGE *The TOE will provide all the functions necessary to support the administrative users and authorized subjects in their management of the configuration data, and restrict these functions from use by unauthorized subjects.*	FMT_MSA_EXP.1 FMT_MTD.3 FMT_MTD_EXP.1	Requiring the TOE to provide adequate functions to manage the TSF configuration data securely satisfies this objective. Since the separation kernel does not support the notion of "authorized roles", creation of explicit requirements, rather than making refinements to the existing FMT components, were necessary. These functional requirements were specifically written to support both static and dynamic management schemes. FMT_MSA_EXP.1 was based on FMT_MSA.1. It requires the configuration data to be the only mechanism through which subjects are designated as authorized subjects. FMT__MTD.3 requires the TSF to perform syntax check on all TSF data. The values that are accepted as valid must fall within the defined range. FMT_MTD_EXP.1 was based on FMT_MTD.1. It disallows modification of the flow policy configuration data.
O.PROTECT *The TOE will provide mechanisms to protect services and exported resources.*	FDP_RIP_EXP.2 FPT_ITI_EXP.1 FPT_RVM.1 FPT_SEP.3	FDP_RIP_EXP.2 requires the TSF to provide a mechanism to guard against unauthorized disclosure of residual information of exported resources. The ST author is responsible to specify the event that invokes this mechanism, i.e., either upon the allocation or upon the deallocation of the exported resources. FPT_ITI_EXP.1 requires the ST author to specify the desired strength of the modification detection mechanism, and the actions to be taken if a modification of the TSF data has been detected. FPT_RVM.1 requires the TSF to enforce the TSP on all services and exported resources. The security domain of a subject includes the services and exported resources that the particular subject is allowed to use. FPT_SEP.3 requires the TSF to enforce separation between the security domains of all subjects in the TSC, thus ensuring that subjects cannot access or manipulate other subject's services and resources in violation of the TSP.
O.RATING_MAINTENANCE *Procedures to maintain the TOE's rating will be documented.*	AMA_AMP_EXP.1	The AMA family of requirements is incorporated into this PP to ensure the TOE developer has procedures and mechanisms in place to maintain the evaluated rating that is ultimately awarded the TOE. These requirements are somewhat related to the ACM family of requirements in that changes to the TOE and its evidence must be managed, but the AMA requirements ensure the appropriate level of analysis is performed on any changes

Objectives from Policies/Threats	Requirements Meeting Objectives	Rationale
		made to the TOE to ensure the changes do not affect the TOE's ability to enforce its security policies. AMA_AMP_EXP.1 requires the developer to develop an assurance maintenance (AM) plan that describes how the assurance gained from an evaluation will be maintained, and that any changes to the TOE will be analyzed to determine the security impact, if any, of the changes that are made. This requirement mandates the developer assign personnel to fulfill the role of a security analyst that is responsible for ensuring the changes made to the TOE will not adversely impact the TOE and that it will continue to maintain its evaluation rating.
O.RECOVERY *Procedures and/or mechanisms will be provided to assure that recovery, such as from system failure or discontinuity, is obtained without a protection compromise.*	FPT_FLS.1 FPT_RCV.2 FPT_RCV.4	FPT_FLS.1 requires the TSF to fail securely, i.e., to preserve a secure state, when failures are detected by the TSF self-tests. FPT_RCV.2 requires the TSF to return to a secure state using automated procedures, i.e., without human intervention, after the occurrence of an ST-defined failure or service discontinuity condition. The ST author is required to fill in the list of failures/service discontinuities to be recovered automatically. If automated recovery is not possible, the TSF is also required to enter a maintenance mode that allows the TOE to return to a secure state. It is assumed that the IT environment provide adequate protection against unauthorized access to the maintenance mode. FPT_RCV.4 requires the TSF to ensure that all security functions that affect secure state can recover to a consistent and secure state if a ST-defined failure scenario is encountered during its execution. The ST author is required to fill in the list of failure scenarios from which the TSF is expected to recover.
O.REFERENCE_MONITOR *The TOE will maintain a domain for its own execution that protects itself and its resources from external interference, tampering, or unauthorized disclosure.*	FPT_RCV.2 FPT_SEP.3	The requirements that implement this objective fall into two categories. The first category requires the TSF to create and maintain separate security domains for its execution. The second category requires the TSF to continue to protect itself even after unexpected interruptions, i.e., to be able to recover to a consistent and secure state. FPT_SEP.3 belongs to the first category. It requires the TSF to maintain three different types of security domains during runtime: 1) a separate domain for information flow control, partition flow control and cryptography SFP enforcement functions, 2) a separate domain for the remainder of the TSF that does not enforce the flow control and cryptography SFPs, and 3) separate domains for the non-TSF portions of the TOE, i.e., the subjects in the TSC. The SFP enforcement functions are the most important functions provided by the TSF, thus it is necessary to separate them from the less-critical portion of the TSF. The separation between the TSF and the non-TSF portion of the TOE is also necessary so that the non-TSF portion cannot interfere with the operation of the TSF. FPT_RCV.2 belongs to the second category. It requires the TSF

Objectives from Policies/Threats	Requirements Meeting Objectives	Rationale
		to enter a maintenance mode that allows the TOE to return to a secure state if automated recovery is not possible. This ensures that the TSF cannot be bypassed even in the event of non-recoverable failures.
O.RESIDUAL_INFORMATION *The TOE will ensure that any data contained in a protected resource is not released when the resource is reallocated.*	FDP_RIP_EXP.2	FDP_RIP_EXP.2 satisfies this objective by requiring that when an exported resource is reallocated, the TSF must ensure that no residual information from the previous allocation is made available via that particular exported resource. Removal of residual information must occur at the point of deallocation or allocation. The ST author needs to make the selection based on the intended use of the TOE.
O.RESOURCE_SHARING *The TOE shall provide mechanisms that mitigate attempts to exhaust TOE resources (e.g., system memory, persistent storage, and processing time).*	FRU_RSA_EXP.1	Requiring the TSF to statically allocate exported resources to partitions as defined by the configuration data satisfies this objective. For separation kernels, the applicable exported resources include system memory and processing time. FRU_RSA_EXP.1 was created because the existing FRU_RSA.1 mandates the allocation limits be based on users and subjects, not partitions. Two iterations of this requirement were used to specify the different allocation rules for system memory and processing time. Allocation limits on system memory are based on the simultaneous memory usage by the partitions at any given time. Allocation limits on processing time are based on the CPU usage by the partitions over a specific time interval. Limits regarding the exhaustion of other exported resources is left to the ST author.
O.SECURE_STATE *The TOE will provide mechanisms to transition the TSF to a secure state during start-up, re-activation of the current flow policy configuration data and activation of a new flow policy configuration data.*	ADV_INI_EXP.1	Abstractly, the TOE consists of two distinct sets of functions: initialization and runtime. Initialization functions are outside the scope of the TSF because they set up the TSF. Initialization functions only execute during start-up and are not relied upon for security enforcement after the TOE is fully initialized. Runtime functions, on the other hand, are relied upon, either directly or indirectly, to correctly enforce the TSP once the TSF reaches a secure state. ADV_INI_EXP.1 requires the TOE's developer to provide an initialization mechanism that brings the TSF to a secure state after the TSF code and data are transferred into memory. The combination of the boot and initialization functions satisfies this objective. ADV_INI_EXP.1 also requires the developer to provide an initialization mechanism to provide restrictive defaults for all security attributes that are not explicitly set by the administrator. <Need to revisit the above rationale after ADV_INI is finalized>
O.SOUND_DESIGN *The TOE will be designed using sound design principles and techniques. The TOE design, design principles and design techniques will be adequately*	ADV_ARC_EXP.1 ADV_FSP.4 ADV_HLD_EXP.4 ADV_INT_EXP.3	This objective is achieved by imposing developmental requirements on the design of the TSF and non-TSF components of the TOE, and on the analysis of the security functions for which strength of function claims are made. For this PP, strength of function claims are made only on cryptographic functions.

Objectives from Policies/Threats	Requirements Meeting Objectives	Rationale
and accurately documented.	ADV_LLD_EXP.2 ADV_RCR.3 ADV_SPM.3 AVA_SOF.1	Rationales for ADV requirements are described in Appendix E. <Rationale for AVA_SOF depends on crypto support>
O.SOUND_IMPLEMENTATION *The implementation of the TOE will be an accurate instantiation of its design.*	ADV_IMP.3 ADV_INT_EXP.3 ADV_LLD_EXP.2 ADV_RCR.3 ALC_DVS.2 ALC_FLR.3 ATE_COV.3 ATE_DPT.2 ATE_FUN.2 ATE_IND.3 AVA_CCA_EXP.2 AVA_VLA.4	This objective is achieved by imposing developmental requirements on the implementation of the TSF and non-TSF components of the TOE to ensure that the TOE implementation is correctly created as specified by the TOE design. Rationales for ADV requirements are described in Appendix E. ALC_DVS.2 requires the developer to describe all security measures they employ to ensure the integrity and confidentiality of the TOE are maintained. In addition to showing the evidence that these security measures are followed during the development and maintenance of the TOE, the developer is also required to justify that these security measures provide the necessary level of protection. Although confidentiality may not be an issue for some TOE implementation, the physical, procedural, and personnel security measures the developer uses provides an added level of assurance that the integrity of the TOE implementation is appropriately maintained. ALC_FLR.3 supports this objective by requiring the developer to track and correct flaws in the TOE, and to provide safeguards that new flaws are not created while fixing the discovered flaws. ATE_COV.3, ATE_DPT.2 and ATE_FUN.2 require the developer to test the TSF and analyze the test coverage as well as the depth of testing. These requirements provides the assurance that the TOE security functional requirements are correctly implemented and that the TOE implementation is a correct instantiation of both high-level design and low-level design. ATE_IND.3 provides added assurance on the rigor of the testing by requiring the evaluator to develop and run their own test suite in addition to re-testing all tests performed by the developer. The correctness of the TOE implementation can be demonstrated by a successful execution of these tests by the evaluator. Requiring the TOE to be assessed for the existence of exploitable covert channels and vulnerabilities also satisfies this objective. AVA_CCA_EXP.2 requires the developer to perform a systematic search for inter-partition covert channels and potential cryptographic key leakage from the cryptographic module. The NSA evaluator is required to confirm the results of the covert channel analysis and to selectively validate the analysis through testing. See O.COVERT_CHANNEL_ANALYSIS for the rationale on why inter-partition covert channels and thorough search for cryptographic key leakage is important. AVA_VLA.4 component to provide the necessary level of

Objectives from Policies/Threats	Requirements Meeting Objectives	Rationale
		confidence that vulnerabilities do not exist in the TOE that could cause the security policies to be violated. AVA_VLA.4 requires the developer to perform a systematic search for potential vulnerabilities in all the TOE deliverables, and to provide a justification that the analysis completely addresses the TOE deliverables. AVA_VLA.4 was refined to require that, in addition to the independent penetration testing and analysis performed by the evaluator, a second set of penetration testing and analysis be independently performed by the NSA evaluator. The two levels of independent testing and analysis helps to ensure that the TOE is resistant to penetration attacks performed by an attacker possessing a high attack potential.
O.TRUSTED_DELIVERY *The integrity of the code of the TOE must be protected during the initial distribution and subsequent updates, and verified to ensure that the on-site version matches the master distribution version.*	ADO_DEL_EXP.2	ADO_DEL_EXP.2 requires the developer to provide cryptographic signature services and cryptographic hashing functions to protect the integrity of the TOE when distributing versions of the TOE to a user's site. ADO_DEL_EXP.2 also requires the developer to use independent channels to deliver the TOE code and to deliver the cryptographic keying materials used to verify the distribution of the code. Cryptographic integrity check mechanisms increase assurance, i.e., only people possessing the correct cryptographic key will be able to view the cleartext checksum of the code.
O.TSF_INTEGRITY *The TOE will be able to verify the integrity of the TSF code and data.*	FPT_TST_EXP.1	FPT_TST_EXP.1 requires the TSF to either verify, or provide the capability for an authorized subject to verify, the integrity of TSF configuration data and TSF executable code loaded in memory. If the TSF software or TSF configuration data is corrupted, the TSF may not correctly enforce its security policies. Besides the TSF configuration data, the ST author is required to specify the testing of other TSF data that the TSF depends on to enforce its security policies.
O.USER_GUIDANCE *The TOE will provide users with the necessary information for secure use of the TOE.*	AGD_USR.1	AGD_USR.1 satisfies this objective by requiring the developer to document the functions, interfaces and warnings available to non-administrative users of the TOE. AGD_USR.1 further requires the developer to describe all user responsibilities and assumptions necessary for secure use of the TOE.
O.VULNERABILITY_ANALYSIS_TEST *The TOE will undergo independent vulnerability analysis and penetration testing by NSA to demonstrate the design and implementation of the TOE does not allow attackers with high attack potential to violate the TOE's security policies.*	AVA_CCA_EXP.2 AVA_MSU.3 AVA_SOF.1 AVA_VLA.4	AVA_CCA_EXP.2 requires both the developer and evaluator to perform a systematic search for inter-partition covert channels and cryptographic key leakage. . See O.COVERT_CHANNEL_ANALYSIS for the rationale on why it is important to perform a thorough search for these covert channels. AVA_MSU.3 satisfies this objective by requiring the developer to provide complete, clear, consistent and reasonable administrator and user guidance documents, and to perform an analysis for any vulnerability that might be caused by unclear documentation. AVA_MSU.3 further requires the evaluator to perform independent testing to check if the provided guidance document would enable an administrator or user, with proper

Objectives from Policies/Threats	Requirements Meeting Objectives	Rationale
		training, to determine if the TOE is configured correctly or incorrectly. <Rationale for AVA_SOF depends on crypto support> AVA_VLA.4 satisfies this objective by requiring the developer 1) to perform a systematic search for vulnerabilities, 2) to document the disposition of the identified vulnerabilities, 3) and to show evidence that the identified vulnerabilities cannot be exploited in the intended environment for the TOE and that the TOE is resistant to obvious penetration attacks. AVA_VLA.4 also requires two levels of independent testing and analysis to help to ensure that the TOE is resistant to penetration attacks performed by an attacker possessing a high attack potential.

7.5 IT Environment Requirements Rationale

109 Each of the environment security objectives identified in section 4 are addressed by one or more security requirements. Table 7.5 below provides the mapping from security requirements to security objectives, as well as a rationale that discusses how the security objective is met. Definitions are provided (in italics) below each security objective so the PP reader can reference these without having to go back to section 4.

Table 7.5 – Mapping of Security Requirements for IT Environment to Objectives

Objectives from Policies/Threats	Requirements Meeting Objectives	Rationale
OE.CHANNELS *If the residual risk from covert channels is a concern, the applications executing on the TOE must be trusted with assurance commensurate with the value of the IT assets protected by the TOE.*	N/A	IT environment requirements that address this objective are outside the scope of this PP. Covert channels allowed to exist on a system are a threat to the assets protected by the system. Assurance must be provided that Trojan horses and other application malware cannot attack IT assets via the covert channels.
OE_PHYSICAL *Physical security will be provided for the TOE by the IT environment commensurate with the value of the IT assets protected by the TOE.*	N/A	IT environment requirements that address this objective are outside the scope of this PP. Computer systems built to conform with this PP may be vulnerable to physical attack such that they are unable to protected their IT assets.

Objectives from Policies/Threats	Requirements Meeting Objectives	Rationale
OE.TRUSTED_FLOWS *If a subject is allowed by the configuration data to cause information flow in violation of the partial ordering of information flows between partitions, that subject must be trusted with assurance commensurate with the value of the IT assets in all partitions to which it has access.*	N/A	IT environment requirements that address this objective are outside the scope of this PP. See rationale for A.TRUSTED_FLOWS.
OE.TRUSTED_INDIVIDUA L *If an individual is allowed to perform procedures upon which the security of the TOE may depend, that individual must be trusted with assurance commensurate with the value of the IT assets.*	N/A	IT environment requirements that address this objective are outside the scope of this PP. See rationale for A.TRUSTED_INDIVIDUAL.

7.6 Explicit Requirements Rationale

110 Explicit components have been included in this protection profile because the Common Criteria requirements were found to be insufficient as stated. This section includes the rationale for using explicit requirements for both the TOE and the IT environment.

7.6.1 Explicit TOE Functional Requirements

Table 7.6 – Rationale for Explicit TOE Functional Requirements

Explicit Component	Rationale
FCS_BCM_EXP.1	The CC does not provide a means of specifying a cryptographic module baseline for implementations developed in hardware, in software, or in hardware/software combinations. FCS_BCM_EXP.1 provides for the specification of the required FIPS certification based on the implementation baseline.
FDP_RIP_EXP.2	FDP_RIP.2 is defined in terms of resource and objects in the CC. Since this PP does not support the “object” abstraction, FDP_RIP_EXP.2 was introduced.
FMT_MSA_EXP.1	As there is no user interface, “authorized subjects” is used in place of an “authorized role.” Sentence structure was simplified for clarity.
FPT_ITI_EXP.1	The words “during transmission” were changed to “whenever the TSF data is transmitted” to indicate that such transmission is not required.

Explicit Component	Rationale
FPT_TST_EXP.1	”TSF” was changed to “TSF’s implementation” to indicate that this requirement applies to the software and not the hardware. As there is no user interface, “authorized subjects” is used in place of “authorized users.” Also modified to allow that the TSF itself may perform this verification. Added to account for the verification of the integrity of the currently running TSF (viz., its executing image)

FRU_RSA_EXP.1(1)
FRU_RSA_EXP.1(2)
	Requirements (1) and (2) were split to allow clarity of presentation, since they involve different metrics (i.e., “simultaneously” vs. “period of time”). As there is no user interface, “partitions” was substituted for the various user expressions.

7.6.2 Explicit TOE Assurance Requirements

Table 7.7 – Rationale for Explicit TOE Assurance Requirements

Explicit Component	Rationale
ADO_DEL_EXP.2	Requirements .3D, and .4C through .7C were added to require the developer to provide documentation for trusted delivery. The requirement of independent channels for delivery of TOE and keying materials provides additional assurance against tampering. Requirement .2E was added to require the evaluator to determine that if the procedures are used as prescribed, trusted delivery can be achieved.
AGD_ADM_EXP.1	Requirements .9C and .10C were added to require that the developer provide guidance on how to use the configuration data generation tool to create configuration data that accurately reflects the user’s intention. FDP_IFF and FDP_IFC require that access to resources be controlled by the TSF at the granularity to which those resources are made available (viz., exported) to subjects. Thus, the TSF provides the ability to enforce least privilege. Requirements .11C and .12C were added to ensure that the developer provide guidance for creating TSF configuration that conforms to the principle of least privilege, and that the configuration data, in fact, enforces least privilege. Requirements .13C through .16C were added to require the developer to document how to generate the configuration data, to convert the TSF code and/or configuration data into a TSF-useable form and to bring the TSF to the initial secure state. Requirements .2E and .3E were added to require the evaluator to determine that the administrator guidance satisfies requirements .9C through .16C.
AMA_AMP_EXP.1	Requirement .1.1C was added to make explicit the requirement to identify the assurance baseline.

Explicit Component	Rationale
AVA_CCA_EXP.2	Requirement .2.1D was refined to only address covert channels between partitions. Requirement .2.2D was added to ensure that a thorough search for potential cryptographic key leakage is performed by the developer. Requirements .2.1E through .2.3E were refined to explicitly require the NSA evaluator to confirm the results of the covert channel analysis and to selectively validate through testing the covert channel analysis.
ADV requirements for TSF	See Appendix E.
ADV requirements for Tools	<TBD>

7.7 Rationale for Strength of Function

111 TBD

7.8 Rationale for Assurance Rating

112 This protection profile has been developed for a U.S. Government high robustness environment. The TOE environment and the value of information processed by this environment (i.e., highly sensitive) establish the need for the TOE to be evaluated at an Evaluated Assurance Level 6 Augmented (EAL6+)[9].

[9] Refer to the "Mutual Recognition of Common Criteria Certificates" section 1.3 to read conditions for the CC certificate to be mutually recognized for PPs with EALs higher than 4.

8. References

[1] Common Criteria for Information Technology Security Evaluation, Part 1: Introduction and General Model, CCIMB-99-031, Version 2.1, August 1999.

[2] Common Criteria for Information Technology Security Evaluation, Part 2I: Security Functional Requirements, CCIMB-99-032, Version 2.1, August 1999.

[3] Common Criteria for Information Technology Security Evaluation, Part 3: Security Assurance Requirements, CCIMB-99-033, Version 2.1, August 1999.

[4] Common Methodology for Information Technology Security Evaluation, Part 2: Evaluation Methodology, CEM-99/045, Version 1.0, August 1999.

[5] National Computer Security Center, Department of Defense Trusted Computer System Evaluation Criteria DoD 5200.28-STD, December 1985.

[6] National Security Telecommunications and Information Systems Security Committee, National Information Systems Security (INFOSEC) Glossary, NSTISSI No. 4009, September 2000.

Appendix A - Acronyms

ANSI	American National Standards Institute
CC	Common Criteria for Information Technology Security Evaluation Version 2.1
COTS	Commercial-Off-The-Shelf
DoD	Department of Defense
EAL	Evaluation Assurance Level
FIPS	Federal Information Processing Standard
IA	Information Assurance
IT	Information Technology
NIST	National Institute of Standards and Technology
PKCS	Public Key Cryptography Standards
PKI	Public Key Infrastructure
PP	Protection Profile
RNG	Random Number Generator
SF	Security Function
SFP	Security Function Policy
SFR	Security Function Requirement
SOF	Strength of Function
ST	Security Target
TOE	Target of Evaluation
TOM	Target of Maintenance
TSC	TSF Scope of Control
TSF	TOE Security Functions
TSFI	TSF Interface
TSP	TOE Security Policy

Appendix B - Cryptographic Standards, Policies, and Other Publications

Standards

ANSI X9.31-1998	American National Standards Institute (ANSI) X9.31-1998 (May 1998), Digital Signatures Using Reversible Public Key Cryptography for the Financial Services Industry (rDSA), [http://webstore.ansi.org/ansidocstore].
ANSI X9.62-1998	American National Standards Institute (ANSI) X9.62-1-1998 (10 Oct 1999), Public Key Cryptography for the Financial Services Industry: the Elliptic Curve Digital Signature Algorithm (ECDSA), (http://webstore.ansi.org/ansidocstore).
FIPS PUB 140-2	National Institute of Standards and Technology, Security Requirements for Cryptographic Modules, Federal Information Processing Standard Publication (FIPS-PUB) 140-2, dated May 25, 2001, [http://cs-www.ncsl.nist.gov/publications/fips/fips140-2/fips1402.pdf].
FIPS PUB 171	National Institute of Standards and Technology, Key Management Using ANSI X9.17, Federal Information Processing Standard Publication (FIPS-PUB) 171, dated April 1992 [http://cs-www.ncsl.nist.gov/publicatins/fips/fips171/fips171.txt].
FIPS PUB 180-2	National Institute of Standards and Technology, Secure Hash Standard, Federal Information Processing Standard Publication (FIPS-PUB) 180-2, dated 1 August 2002, [http://cs-www.ncsl.nist.gov/publications/fips/fips180-2/fips180-2.pdf].

Appendix C – Rationale for IFC/IFF Requirements

113 The requirements for IFC and IFF need to include both a partition level policy and a subject/resource level policy for several reasons:

1. This PP should be specific to separation kernels. This is not intended to be a generic security kernel PP. Stating IFC and IFF as generic access control mechanisms would allow any security kernel to be evaluated under this PP.
2. Covert channel requirements should apply at the partition level, but not within partitions. If IFC/IFF were written without concern for partitions, then it would be inconsistent to state covert channel requirements at the partition level.
3. Requirements for Principle of Least Privilege appropriate for high assurance systems should apply at both the partition level and at more granular levels. For example, suppose there is one subject per partition, and the partition includes several exported resources. In this case, the subject should have only (viz., the least) modes of access to each of those resources that it requires, as opposed to a blanket (e.g., maximal) access to all of the resources in the partition. This restriction may be difficult to express or understand if IFC/IFF does not articulate requirements at the partition as well as the resource level.

114 With respect to this PP, the TSF creates the subject and partition abstractions from the internal resources available on a single processor, and exports at its interface certain resources to subjects. It provides separation between subjects and partitions by controlling which of those exported resources each subject may access. Even if there is only one subject in a partition (there is a program there, regardless of whether or not the implementation uses the word subject, or partition for that matter), the TSF must still ensure that the resources that the subject can access (viz., via its address space) are only those resources that the TSF has exported to it: it is the premise of separation that the TSF must always know what resources to allow or deny to any subject. It would be circular to say that, since separation is provided, the TSF need not provide access control to selected resources exported to subjects.

115 A high assurance TSF must be able to apply least privilege to those resources it exports (as well as being structured to enforce least privilege internally). A subject's address space is defined by a set of exported resources and the access modes granted to that subject for those resources. Least privilege regarding exported resources should not be difficult for any implementation of a separation kernel, since the configuration data will always identify the composition of each subject's address space, which does not change dynamically during runtime. Least privilege is an important notion and should be implemented in any secure system, although in the "minimal configurations," (see Section 2.1) the requirement for least privilege may be met implicitly.

116 Regarding nomenclature, "partition" and "subject" should be viewed as orthogonal abstractions. "Partition," as discerned from its mathematical genesis, provides for a set-theoretic grouping of system entities, whereas "subject" allows us to reason about the individual active entities of a system. In this view, it is not consistent to say that a partition (a collection, containing at least

one element) is a subject (an active element). Furthermore, if a partition were a subject, and there were also resources associated with the partition, then we would be back to where we started, with a set of subjects and resources associated with a partition. If there are no resources associated with the degenerative partition/subject, then all exported resources would be outside of that partition, and the subject would not have any address space. In this case, how could a subject run without any stack or code?

117 It is not required that the implementation must perform an explicit assignment of subject or resource to partition; for example, it is allowed for the partition to contain only one subject and for the subject ID and the partition ID to be the same; note that this is not the same as saying that the partition is a subject.

Appendix D – TSF Data Description

118 There are various types of TSF Data, for example: internal data structures, configuration data, and TSF-generated data. Configuration data includes flow policy and non-flow policy data. Some or all configuration data may be imported from the IT environment during system initialization. The TSF generates some data, such as audit records and digital signatures. The TSF may export certain TSF Data, including generated data, configuration data, and other implementation-dependent TSF Data.

119 Examples of TSF data are, Internal TSF Structures, Configuration Data and TSF-Generated Data:

A. Internal TSF Structures
   1. Hardware registers
   2. Software data structures

B. Configuration Data
   1. Flow Policy Configuration Data
      a. Information Flow Configuration Data
      b. Partition Flow Configuration Data
   2. Non-Flow Policy Configuration Data
      c. Audit Configuration Parameters
      d. General Configuration Parameters
         i. Clock Settings
         ii. Self-Test Periods

B. TSF-Generated Data
   1. Subject and resource policy-enforcement attributes
   2. Audit Output (e.g., audit records)
   3. Clock Output

Appendix E – Explanatory Material for Explicit Class ADV Requirements

E.1 Rationale for Class ADV: Development

Class ADV: Development

Editor Note: ***Document version 0.1; 15 January 2004.***

Editor Note: ***I suggest when reading this for the first time, you start with the families that already existed, then ARC because, although it is new, it ties in with them. CMP and IFA address information related to the composition issue; we discovered a need for them in HLD, but they really are distinct from the others.***

Editor Note: ***This is the proposed leveling of ADV requirements using the new components.***

Assurance Class	Assurance Family	Assurance Components by Evaluation Assurance Level						
		EAL1	EAL2	EAL3	EAL4	EAL5	EAL6	EAL7
Development	ADV_ARC				**1**	1	1	1
	ADV_CMP		**1**	**2**	2	2	2	2
	ADV_FSP	**1**	**2**	**3**	3	**4**	**5**	**6**
	ADV_HLD		**1**	**2**	2	**3**	**4**	4
	ADV_IFA							
	ADV_IMP				**1**	1	**2**	**3**
	ADV_INT					**2**	**3**	4
	ADV_LLD				**1**	**2**	**3**	**4**
	ADV_RCR		**1**	1	**2**	2	2	**3**
	ADV_SPM				**1**	**3**	3	3

120 The development class encompasses five families of requirements for representing the TSF at

various levels of abstraction from the functional interface to the implementation representation. These families include the description of mechanism-oriented Security Functional Requirements (SFRs) as well as those that are architecture-oriented (specifically, FPT_SEP and FPT_RVM). The development class also includes a family of requirements for a correspondence mapping between the various TSF representations to aid in the determination that the levels of decomposition are mutually supportive. There are two families included to support the activities required in order to integrate ("compose") evaluated TOEs that include those families in a way that facilitates the determination of the trustworthiness of the integrated system. In addition, there is a family of requirements for a TSP model, and for correspondence mappings between the TSP, the TSP model, and the functional specification. Finally, there is a family of requirements on the internal structure of the TSF, which covers aspects such as modularity, layering, minimisation of complexity, and principle of least privilege.

121 Figure E-1 shows the families within this class, and the hierarchy of components within the families.

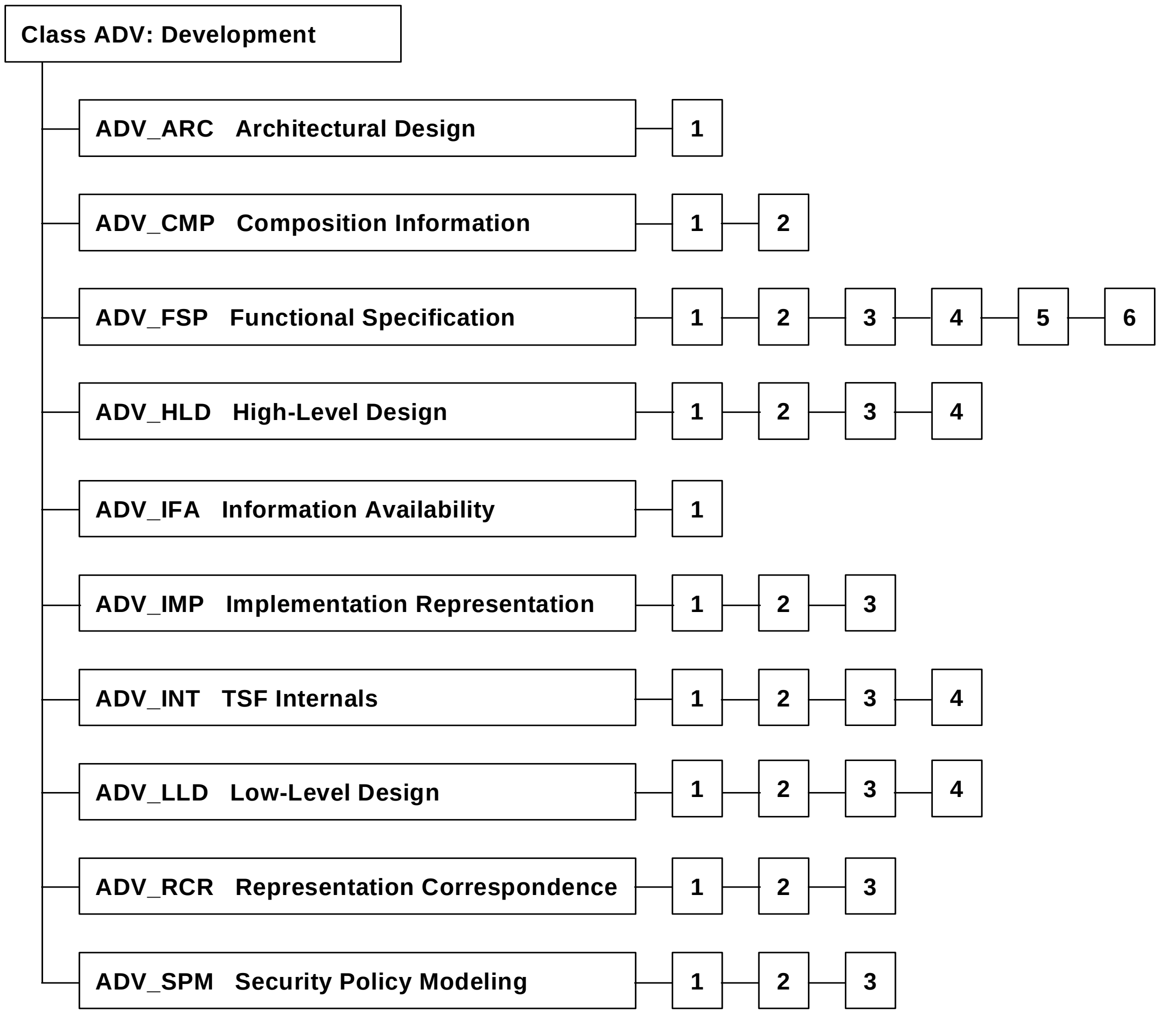

Class ADV: Development
ADV_ARC Architectural Design
1
ADV_CMP Composition Information
1 2
ADV_FSP Functional Specification
1 2 3 4 5 6
ADV_HLD High-Level Design
1 2 3 4
ADV_IFA Information Availability
1
ADV_IMP Implementation Representation
1 2 3
ADV_INT TSF Internals
1 2 3 4
ADV_LLD Low-Level Design
1 2 3 4
ADV_RCR Representation Correspondence
1 2 3
ADV_SPM Security Policy Modeling
1 2 3

Figure E-1 Development Class Decomposition

122 The paradigm evident for these families is one of a functional specification of the TSF in terms of its interfaces (describing *what* the TSF does), decomposing the TSF into subsystems (describing *how* the TSF accomplishes its functions at a higher level), decomposing the subsystems into modules (describing *how* the TSF accomplishes its functions at an algorithmic level, and providing a guide for the review of the implementation representation), and showing the implementation of the modules. All levels of decomposition are used in determining the completeness and accuracy of all other levels, ensuring that the levels are mutually supportive. The requirements for the various TSF representations are separated into different families, to allow the PP/ST author to specify which subset of the TSF representations are required. The level chosen will dictate the assurance desired/gained.

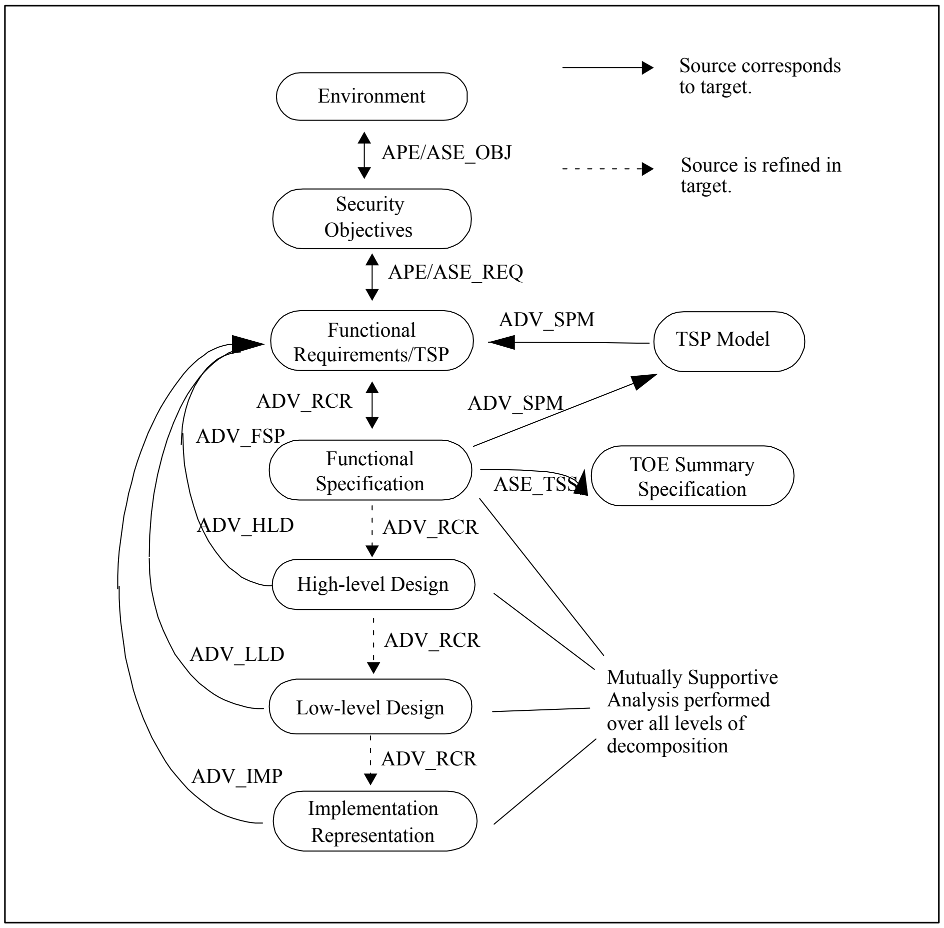

Source corresponds to target.
Source is refined in target.
Environment
APE/ASE_OBJ
Security Objectives
APE/ASE_REQ
Functional Requirements/TSP
ADV_SPM
TSP Model
ADV_RCR
ADV_SPM
ADV_FSP
Functional Specification
ASE_TSS
TOE Summary Specification
ADV_HLD
ADV_RCR
High-level Design
ADV_RCR
ADV_LLD
Low-level Design
Mutually Supportive Analysis performed over all levels of decomposition
ADV_RCR
ADV_IMP
Implementation Representation

Figure E-2 Relationships between TOE Representations and Requirements

Editor Note: ***We will need to modify (eventually) this figure and the references to it, once families are agreed. Currently para 304 indicates INT, ARC, and IFA are not pictured, but may want to change this if we can figure a way to represent them here. (ADV_ARC contains FSP- and HLD-type information for the FPT_SEP and FPT_RVM functional requirements. ADV_CMP contains FSP-type***

information for calls out from the TSF. ADV_IFA contains information to support ADV_CMP by use of third parties.)

123 Figure E-2 indicates the relationships between the various TSF representations and the objectives and requirements that they are intended to address. As the figure indicates, the APE and ASE classes define the requirements for the correspondence between the functional requirements and the security objectives as well as between the security objectives and the TOE's anticipated environment. Class ASE also defines requirements for the correspondence between both the security objectives and functional requirements and the TOE summary specification.

124 The requirements for all other correspondence shown in Figure E-2 are defined in the ADV class. The ADV_SPM family defines the requirements for correspondence between the TSP and the TSP model, and between the TSP model and the functional specification. The ADV_RCR family defines the requirements for mappings from higher-level representations to "adjacent" lower-level representations (e.g., functional specification to high-level design) to facilitate the analysis of completeness and accuracy at each level of decomposition.

125 Finally, each assurance family specific to a TSF representation (i.e., ADV_FSP, ADV_ARC, ADV_HLD, ADV_LLD and ADV_IMP) defines requirements relating that TSF representation to the functional requirements, the combination of which helps to ensure that the TOE security functional requirements have been addressed. All decompositions must completely and accurately reflect all other decompositions (i.e., be mutually supportive). Assurance relating to this factor is obtained by analysis of each of the levels of decomposition and comparison of the details with other levels of decomposition (in a recursive fashion) while the analysis of a particular level of decomposition is being performed.

126 ADV_CMP requires that calls out of the TSF (that is, functionality the TSF depends on in order to perform its functions) be documented and traced to the TSFI that, when invoked, will in turn cause the call out of the TSF to occur. While the activities during product evaluation can only determine accuracy and completeness of such documentation (and not their security effect), the information is made available (along with that in the ADV_IFA family) to allow products to be integrated and analyzed from a security perspective.

127 The ADV_INT family is not represented in this figure, as it is related to the internal structure of the TSF, and is only indirectly related to the process of refinement of the TSF representations. Similarly, the ADV_ARC family is not represented in the figure because it relates to the architectural soundness, rather than representation, of the TSF. Finally, ADV_IFA families is not included as it relates to the provision of information to be used in analysis or activities outside of the scope of the Common Criteria evaluation paradigm.

Application note

128 The TOE security policy (TSP) is the set of rules that regulate how resources are managed, protected and distributed within a TOE, expressed by the TOE security functional requirements. The developer is not explicitly required to provide a TSP, as the TSP is expressed by the TOE security functional requirements, through a combination of security function policies (SFPs) and the other individual requirement elements.

129 The TOE security functions (TSF) are all the parts of the TOE that have to be relied upon for enforcement of the TSP. The TSF includes both functions that directly enforce the TSP, and also those functions that, while not directly enforcing the TSP, contribute to the enforcement of the TSP in a more indirect manner.

130 Several important concepts were used in the development of the components of the ADV families. These concepts, while introduced briefly here, are explained more fully in the application notes for the families.

131 One over-riding notion is that, as more information becomes available, greater assurance can be obtained that that the security functions are 1) correctly implemented; 2) cannot be compromised; and 3) cannot be bypassed. This is done through the verification that the documentation is correct and consistent with other documentation, and by providing information that can be used to ensure that the testing activities (both functional and penetration testing) are comprehensive. This is reflected in the leveling of the components of the families. In general, components are leveled based on the amount of information that is to be provided (and subsequently analyzed).

132 While not true for all TOEs, it is generally the case that the TSF (which enforces the TSP, meaning the portions of the system that implements the SFRs) is sufficiently complex that there are portions of the TSF that deserve more intense examination than other portions of the TSF. Determining those portions is unfortunately somewhat subjective, thus terminology and components have been defined such that as the level of assurance increases, the responsibility for determining what portions of the TSF need to be examined in detail shifts from the developer of the TOE to the evaluator of the TOE. To aid in expressing this concept, the following terminology is introduced.

133 All portions of the TSF are *security relevant*, meaning that they must preserve the security of the system as expressed by the SFRs. If a part of the TSF plays a role in implementing any SFR on the system (with the exception of FPT_SEP and FPT_RVM, as detailed in the next paragraph), then that interface is termed *security enforcing*. Such requirements are not limited to the access control requirements, but refer to any functionality provided by one of the SFRs contained in the ST (with exceptions for FPT_SEP and FPT_RVM). It should be noted that the definition of “plays a role in” is impossible to express quantitatively, thus leading to the distinction of what is security enforcing and what is merely *security supporting*. Security-supporting functionality is trusted to preserve the security, both by operating correctly and not being subject to corruption. The distinction between security-supporting and security-enforcing functionality is defined through the requirements in the components and the associated methodology for the components in the ADV families dealing with TOE representation.

134 FPT_SEP and FPT_RVM are SFRs that require a different type of analysis from other SFRs. These requirements are architecturally related, and their implementation (or lack thereof) is not easily (or efficiently) testable at the TSFI. From a terminology standpoint, although implementation (and the associated analysis) of FPT_SEP and FPT_RVM is critical to the trustworthiness of the system, portions of the TSF whose sole security relevance is applicable to meeting the FPT_SEP and FPT_RVM requirements will be termed security supporting.

135 It is important to point out that in terms of the assurance to be provided, the analysis for security supporting functionality (for instance, properties of a system that support FPT_SEP and

FPT_RVM) is no less important than that for security enforcing functionality. There is a difference in analysis of these two types of functionality, however, in that the security-enforcing functionality is more or less directly visible and relatively easy to test, while security-supporting functionality requires varying degrees of analysis on a much broader set of functionality. Further, the depth of analysis for security-supporting functionality will vary depending on the design of the system. The ADV families are constructed to address this by have a separate family (ADV_ARC) devoted to analysis of the FPT_SEP and FPT_RVM requirements, while the other families are concerned with analysis of SFRs other than FPT_SEP and FPT_RVM.

136 Even in cases where different descriptions are necessary for the multiple levels of abstraction, it is not absolutely necessary for each and every TSF representation (with the exception of the implementation representation) to be in a separate document. Indeed, it may be the case that a single document meets the documentation requirements for more than one TSF representation, since it is the information about each of these TSF representations that is required, rather than the resulting document structure. In cases where multiple TSF representations are combined within a single document, the developer should indicate which portions of the documents meet which requirements. The exception is that the implementation representation has to be independent from the low-level design document in order to provide a clear and un-ambiguous low-level design for the TSF.

137 Three types of specification style are mandated by this class: informal, semiformal and formal. The functional specification, high-level design, low-level design and TSP models will be written using one or more of these specification styles. Ambiguity in these specifications is reduced by using an increased level of formality; however, formal specification of certain decompositions is currently beyond the state of the art and is not included.

138 An informal specification is written as prose in natural language. Natural language is used here as meaning communication in any commonly spoken tongue (e.g. Dutch, English, French, German). An informal specification is not subject to any notational or special restrictions other than those required as ordinary conventions for that language (e.g. grammar and syntax). While no notational restrictions apply, the informal specification is also required to provide defined meanings for terms that are used in a context other than that accepted by normal usage.

139 The difference between semiformal and informal documents is only a matter of formatting/presentation: a semiformal model includes such things as an explicit glossary of terms, a standardized presentation format, etc. A semiformal specification is written in a restricted syntax language and is typically accompanied by supporting explanatory (informal) prose. The restricted syntax language may be a natural language with restricted sentence structure and keywords with special meanings, or it may be diagrammatic (e.g. data-flow diagrams, state transition diagrams, entity-relationship diagrams, data structure diagrams, and process or program structure diagrams). Whether based on diagrams or natural language, a set of conventions must be supplied to define the restrictions placed on the syntax. Inclusion in the glossary explicitly identifies the words that are being used in a precise and constant manner; similarly, the standardised format implies that extreme care has been taken in methodically preparing the document in a manner that maximizes clarity.

140 A formal specification is written in a notation based upon well-established mathematical concepts, and is typically accompanied by supporting explanatory (informal) prose. These

mathematical concepts are used to define the syntax and semantics of the notation and the proof rules that support logical reasoning. The syntactic and semantic rules supporting a formal notation should define how to recognise constructs unambiguously and determine their meaning. There needs to be evidence that it is impossible to derive contradictions, and all rules supporting the notation need to be defined or referenced.

E.2 Rationale for ADV_ARC_EXP

Architectural Design (ADV_ARC)

Editor Note: ***This version of ADV acknowledges the inherent difference in nature between the pseudo-architectural requirements (FPT_SEP and FPT_RVM) and the purely functional requirements (the others in Part 2). This difference is reflected by treating the two separately within the FSP, HLD, and LLD families. Consequently, if ADV_ARC is not included when FPT_SEP and FPT_RVM are included (in either the PP or ST) then they will not be analyzed in doing the ADV_FSP, HLD, and LLD work units, and this would not be a good thing. Therefore, there should be a dependency in Part 2 for FPT_SEP and FPT_RVM on this component.***

Objectives

141 The architectural design of the TOE is related to the information contained in other decomposition documentation (functional specification, high-level design, low-level design) provided for the TSF, but presents the design in a manner that supports the argument that the TSP cannot be compromised (FPT_SEP) and that it cannot be bypassed (FPT_RVM). The objective of this component is for the developer to provide an architectural design and justification associated with the integrity and non-bypassability properties of the TSF at the level of detail of the most detailed evidence presented for the TOE.

Component levelling

142 There is only one component in this family.

Application note

143 FPT_SEP and FPT_RVM are distinct from other SFRs because they largely have no directly observable interface at the TSF. Rather, they are properties of the TSF that are achieved through the design of the system, and enforced by the correct implementation of that design. Because of their pervasive nature, the material needed to provide the assurance that these requirements are

being achieved is better suited to a presentation separate from the design decomposition of the TSF as embodied in ADV_FSP, ADV_HLD, and ADV_LLD. This is not to imply that the architectural design called for by this component cannot reference or make use of the design composition material; but it is likely that much of the detail present in the decomposition documentation will not be relevant to the argument being provided for the architectural design document.

144 The architectural design document consists of two types of information. The first is the design information for the entire TSF related to the FPT_SEP and FPT_RVM requirements. This type of information, like the decompositions for ADV_HLD and ADV_FSP, describes how the TSF is implemented. The description, however, should be focused on providing information sufficient for the reader to determine that the TSF implementation is likely not to be compromised, and that the TSP enforcement mechanisms (that is, those that are implementing SFRs other than FPT_SEP and FPT_RVM) are likely always being invoked. The level of this description is commensurate with the most detailed level of decomposition required for the TSF. For example, the architectural design document for a TOE with only an ADV_FSP.1 component requirement and no ADV_HLD.1 requirement would contain primarily configuration information that is visible from the external interface (protections on files that comprise the TSF, for instance). On the other hand, a TOE on which was levied an ADV_HLD component would have an architectural design document that contained more information on the internal design of the self-protection mechanisms (for instance, the memory management architecture).

145 The nature of the FPT_SEP requirement lends itself to a design description much better than FPT_RVM. For FPT_SEP, mechanisms that implement domain separation (e.g., memory management, protected processing modes provided by the hardware, etc.) can be identified and described. However, FPT_RVM is concerned with interfaces that bypass the enforcement mechanisms. In most cases this is a consequence of the implementation, where if a programmer is writing an interface that accesses or manipulates an object, it is that programmer's responsibility to use interfaces that are part of the TSP enforcement mechanism for the object and not to try to "go around" those interfaces. For this reason, the information in the architectural design document for FPT_SEP is somewhat different than that for FPT_RVM, in that for FPT_SEP it is divided into a design description (for the mechanisms) and a justification (analysis) that the mechanisms achieve the FPT_SEP functionality. For FPT_RVM, the information with respect to the mechanisms and justification are not separate because of the nature of the presentation that needs to be made.

146 For FPT_SEP, the design description should cover how user input is handled by privileged-mode routines; what hardware self-protection mechanisms are used and how they work (e.g., memory management hardware, including translation lookaside buffers); how software portions of the TSF use the hardware self-protection mechanisms in providing their functions; and any software protection constructs or coding conventions that contribute to meeting FPT_SEP. The questions that should be answered by this description include:

> What basic mechanisms are used to ensure an untrusted program cannot access a TSF's address space?
>
> For each interface, what prevents that interface from being used to access TSF programs and data?

What protections are there for TSF configuration data (TSF binaries, startup files, etc.)?

What programmatic protections are in place (buffer overflow prevention)?

147 In addition to the descriptive information indicated in the previous paragraphs, the second type of information an architectural design document must contain for FPT_SEP is a justification that the FPT_SEP requirement is being met. This is distinct from the description, and presents an argument for why the design presented in the description is sufficient.

148 For FPT_SEP, the justification should cover the possible modes by which the TSF could be compromised, and how the mechanisms implemented in response to FPT_SEP counter such compromises. The vulnerability analysis might be referenced in this section.

149 For FPT_RVM, the property that the security policies cannot be bypassed applies not only to the access control policies, but to all other security functionality as well. That is, the design description should cover resources that are protected under the SFRs (usually FDP_* components) and functionality (e.g., audit) that is provided by the TSF. The description should also identify the interfaces that are associated with each of the resources or the functionality; this might make use of the information in the FSP. This description should also describe any design constructs, such as object managers, and their method of use. For instance, if routines are to use a standard macro to produce an audit record, this convention is a part of the design that contributes to the non-bypassability of the audit mechanism. It's important to note that "non-bypassability" in this context is not an attempt to answer the question "could a part of the TSF implementation, if malicious, bypass a TSP mechanism", but rather it's to document how the actual implementation does not bypass the mechanisms implementing the TSP. This description typically will include an interface-by-interface analysis showing how the interface either doesn't bypass an appropriate TSF mechanism, or it doesn't use TSF security services and therefore does not play a role in implementing the FPT_RVM requirement.

150 The detailed analysis demonstrates that whenever a resource protected by an SFR is accessed, the protection mechanisms of the TSF are invoked (that is, there are no "backdoor" methods of accessing resources that are not identified and analyzed as part of the ADV_FSP/ADV_HLD/ADV_LLD analysis, depending on the level of evidence supplied). Similarly, the description demonstrates that a function described by an SFR is always provided where required. For example, if the FCO_NRO family were being used the description should demonstrate that all interfaces either 1) do not deal with transmitting the information identified in the FCO_NRO component included in the ST, or 2) invoke the mechanism(s) described by the decomposition documentation.

Dependencies for ADV_ARC_EXP.1

FPT_SEP.1 TSF Domain Separation

FPT_RVM.1 Non-bypassability of the TSP

E.3 Rationale for ADV_CMP_EXP

Composition information (ADV_CMP)

Objectives

151 This family and the ADV_IFA family are intended to be used to provide information to those persons responsible for integrating, or assessing the integration of, the TOE with other IT components in manner that provides confidence that the composed system preserves the security functions of the TOE. The issue of integrating an evaluated component with other components is commonly called the composition problem. There are two distinct aspects to this issue: composition "in the large" and composition "in the small". While there may be some subjectivity in categorizing the composition of specific sets of TOEs, in general composition in the large refers to composing relatively complete products or systems (an OS, firewall, or secure router) into a larger system. In contrast, composition in the small refers to integrating a TOE whose TSF has dependencies on the IT environment into that environment. Integrating an evaluated database onto an OS platform and integrating an OS onto a hardware platform are both example of composition in the small.

152 This family and the ADV_IFA family are targeted at the problem of composition "in the small." The are two aspects to this problem; those for the "base TOE" (covered in ADV_IFA) and those for the "dependant TOE" (covered in this family). A base TOE is a TOE whose TSF may or may not have dependencies on the IT environment, but is expected to have other TOEs combined with it in an operational environment. An operating system would be an example of such a TOE. A dependant TOE is a TOE whose TSF has dependencies on the IT environment, which may be implemented in one or more base TOEs. A database or firewall application would be an example of a dependant TOE.

153 There are no requirements or actions specified that an integrator would follow to actually compose the various components, nor evaluate the result of that composition effort. Rather, the purpose of this family is to ensure that the information to perform those two activities is provided by the TOE developer to the system integrator, persons wishing to analyze a composed system, or other parties that need that information.

Component levelling

154 Components in this family are levelled based on the amount of information required to be provided by the developer.

Application note

155 Throughout this family the term "system integrators" is used to refer to the end users of the information produced as a result of meeting the requirements of this family. It should be noted that system integrators are not the only consumers of this information. Members of teams

responsible for assessing the overall security of a system in which components have been composed would also find this information key to providing them with a measure of assurance that the security mechanisms have not be compromised (or the extent to which the mechanisms may have been compromised) as a result of the integration activity.

156 It is important to note that the term "IT environment" is used throughout this discussion in a general sense, and not in the sense of the IT Environment section of a PP or ST (although the two are obviously related). In this family the objective is define all interactions initiated by the TSF to entities outside of the TSF. Since the TSF may use or be affected by the results of such interactions, they are all security relevant and need to be described. Note that such interactions are not limited to those invoking functionality associated with Requirements on the IT Environment contained in a PP or ST.

157 The crux of the composition issue is the desire to evaluate a component once, then use that component in a variety of systems without requiring further evaluation of the entire system, while maintaining a base level of trust in the overall (integrated) system. In practice, this desire does not seem achievable in the general sense for several reasons:

a) The evaluated version of the component does not include functionality that is used in the composed system, introducing unknown interactions between that "new" functionality and the evaluated component;

b) The interactions between the evaluated component and other components in the system are not documented, which may leave undiscovered interfaces that can be used to compromise the integrity of the composed system; and

c) A component in a system may be privileged with respect to security policies enforced by other components in the system, thus creating a potential for the privileged component to compromise the integrity of the other components in the system.

158 One approach to alleviating the composition problem is to in a sense remove "composition" from consideration. In this approach, one would specify the entire end system in terms of an overall architecture and individual components, providing enough detail so that the end system would meet its functional goals when the components were assembled, and the overall security policy of the system would be preserved. There are two issues with this approach: first, it does not allow for arbitrary composition of evaluated products; and second, it requires extremely detailed specifications in order to ensure that the security properties are preserved once the components are "composed."

159 Although the composition problem cannot otherwise be fully addressed, information pertaining to evaluated products can be made available that mitigate the risks in composing systems and shorten the assessment time necessary to determine that the level of assurance provided by the composed system is adequate. Using this approach, this family specifies documentation requirements on the developer of the TOE that will mandate such documentation be produced and be made available to integrators. Although there will still be issues when composing

evaluated TOEs with non-evaluated TOEs, composing evaluated TOEs in a trustworthy fashion should be much easier.

160 TOEs whose TSF interacts with the IT environment outside of the TSF invoke functions provided by the IT environment in performing the TOE Security Functions (e.g., system calls, communication with the IT environment over a network connection with a standard or custom network protocol). In these cases, those invoked interfaces, while not TSFI, need to be described for those charged with configuring the TOE for end users. The rationale for requiring this documentation is to aid integrators of the TOE and the underlying system to determine what interfaces in the underlying system might have adverse effects on the TOE, and to provide information to analysts charged with evaluating the trustworthiness of the composition of this TOE with another TOE. Additionally, evaluators of the TOE will be able to gain assurance that all requirements on the IT environment are specified in the ST by analyzing this information.

161 In deciding what interfaces need to be documented for these components, it is important to distinguish the set of usable interfaces as compared to the set of interfaces that may be used in one particular environment. The easiest example is a TOE that runs on (and therefore has dependencies on) both BSD-based Unix systems as well as System V-based Unix systems. If it makes different system calls depending on the type of Unix system upon which it is running, the union of both sets of system calls are what needs to be identified and described to meet the ADV_CMP.1.1C and ADV_CMP.2.1C element.

162 The amount of information presented in the composition information should be commensurate with that provided in the functional specification. This is addressed by the two components in this family. The ADV_CMP.1 component should be used when HLD.1 and FSP.1 are included in the ST, while the requirements for the ADV_CMP.2 component are more commensurate with those in HLD.2 and FSP.2.

Dependencies for ADV_CMP_EXP.1

ADV_FSP_EXP.1 Descriptive Functional Specification

ADV_HLD_EXP.1 Descriptive High-Level Design

E.4 Rationale for ADV_FSP_EXP

Functional specification (ADV_FSP)

Objectives

163 The functional specification is a description of the user-visible interface to the TSF. It is an instantiation of the TOE security functional requirements. The functional specification has to completely address all of the user-visible TOE security functional requirements.

Component levelling

164 The components in this family are levelled on the basis of the degree of detail provided for the external interfaces to the TSF, and the degree of formalism required of the functional specification.

Application note

165 A description of the TSF interfaces (TSFI) provides fundamental evidence on which assurance in the TOE can be built. This assurance is gained primarily through testing of the TSF through its TSFI. The more complete and accurate the description, the more assurance can be gained through the testing activities. This concept is reflected in the required documentation from the developer as specified in the various hierarchical components of this family.

166 Fundamentally, the functional specification provides a description of what the TSF provides to users (as opposed to the high-level design and low-level design, which provide a description of how the functionality is provided). Further, the functional specification provides this information in the form of interface (TSFI) documentation.

167 In order to identify the software interfaces to the TSF, the parts of the TOE that make up the TSF must be identified. This identification is formally a part of HLD analysis, but is implicitly performed (through identification and description of the TSFI) by the developer in cases where ADV_HLD is not included in the assurance package. In this analysis, a portion of the TOE is considered to be in the TSF under two conditions:

a) The software contributes to the satisfaction of security functionality specified by a functional requirement in the ST. This is typically all software that runs in a privileged state of the underlying hardware, as well as software that runs in unprivileged states that performs security functionality.

b) The software used by administrators in order to perform security management activities specified in the guidance documentation. These activities are a superset of those specified by any FMT_* functional requirements in the ST.

168 Identification of the TSFI is a complex undertaking. The TSF is providing services and resources, and so the TSFI are interfaces to the security services/resources the TSF is providing. This is especially relevant for TSFs that have dependencies on the IT environment, because not only is the TSF providing security services (and thus exposing TSFI), but it is also using services of the IT environment. While these are (using the general term) interfaces between the TSF and the IT environment, they are not TSFI. Nonetheless, it is vital to document their existence to integrators and consumers of the system, and thus documentation requirements for these interfaces are specified in ADV_CMP.

169 This concept (and concepts to be discussed in the following paragraphs) is illustrated in the following figure.

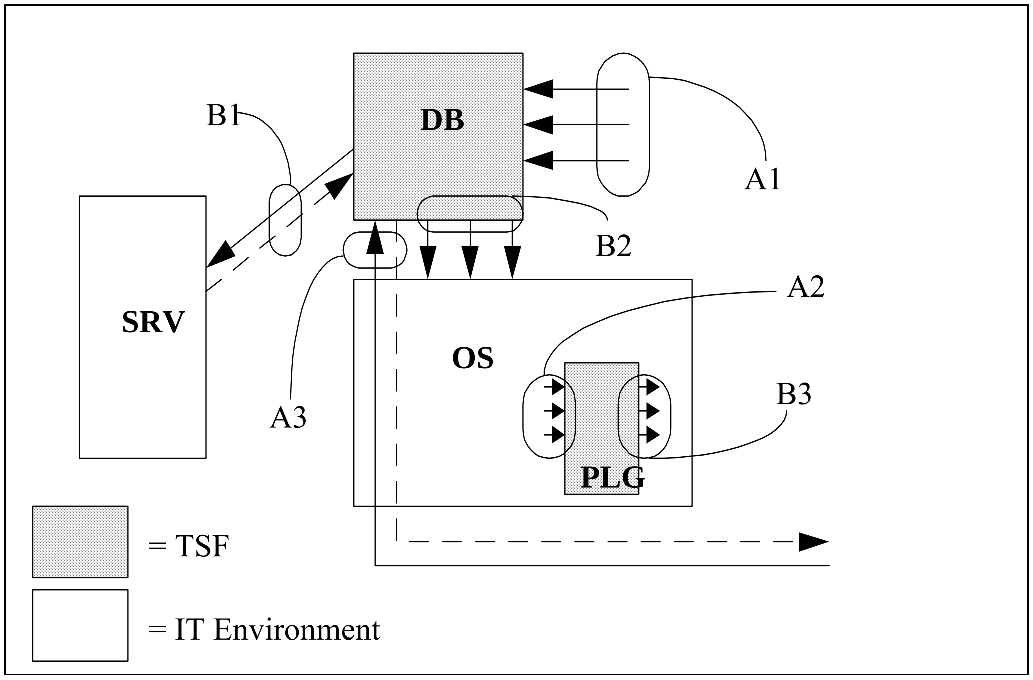


Figure E-3 Interfaces in a DBMS System

170 Figure E-3 illustrates a TOE (a database management system) that has dependencies on the IT environment. The shaded boxes represent the TSF, while the unshaded boxes represent IT entities in the environment. The TSF comprises the database engine and management GUIs (represented by the box labelled "DB") and a kernel module that runs as part of the OS that performs some security function (represented by the box labelled "PLG"). The TSF kernel module has entry points defined by the OS specification that the OS will call to invoke some function (this could be a device driver, or an authentication module, etc.). The key is that this pluggable kernel module is providing security services specified by functional requirements in the ST. The IT environment consists of the operating system (represented by the box labelled "OS") itself, as well as an external server (labelled SRV). This external server, like the OS, provides a service that the TSF depends on, and thus needs to be in the IT environment. Interfaces in the figure are labelled Ax for TSFI, and Bx for interfaces to be documented in ADV_CMP. Each of these groups of interfaces is now discussed.

171 Interface group A1 represents the prototypical set of TSFI. These are interfaces used to directly access the database and its security functionality and resources.

172 Interface group A2 represent the TSFI that the OS invokes to obtain the functionality provided by the pluggable module. These are contrasted with interface group B3, which represent calls that the pluggable module makes to obtain services from the IT environment.

173 Interface group A3 represents TSFI that "pass through" the IT environment. In this case, the DBMS communicates over the network using a proprietary application-level protocol. While the IT environment is responsible for providing various supporting protocols (e.g., Ethernet, IP, TCP), the application layer protocol that is used to obtain services from the DBMS is a TSFI and

must be documented as such. The dotted line indicates return values/services from the TSF over the network connection.

174 Non-TSFI interfaces pictured are labelled Bx. Interface group B1 is the most complex of these, because the architecture of the system and environmental assumptions and conditions will drive its analysis. In the first case, assume that, either through an environmental assumption or an IT environmental requirement, the network link between the DB and SRV is protected (it could be on a separate subnet, or it could be protected by a firewall such that only the DB could connect to the port on the SRV) such that only the DB has access to the SRV. In this case, the interface needs to be documented in the composition guidance (ADV_CMP) only, since untrusted users are unable to gain access.

175 However, consider the case where SRV is now just "somewhere on the network", and now the port that the DB opens up to communicate with the SRV is "exposed" to untrusted users. In this case, while the interface presented by the DB (the TSF) still only needs to be documented in ADV_CMP, additional considerations with respect to vulnerabilities may need to be documented as part of the AVA_VLA activity because of this exposure. In particular, since the TSF is receiving (and trusting) data from this external source, some consideration will have to be given (based on the desired assurance) to "man-in-the-middle" attacks.

176 In the course of performing its functions, the DB will make system calls down to the OS. This is represented by interface group B2. While these calls are not part of the TSFI, they are an interface that needs to be documented in the ADV_CMP.

177 Interface group B3, mentioned previously in connection with interface group A2, is similar to interface group B2 in that these are calls made by the TSF to the IT environment to perform services for the TSF.

178 Having discussed the interfaces in general, the types of TSFI are now discussed in more detail. This discussion categorizes the TSFI into the two categories mentioned previously: TSFI to software directly implementing the SFRs, and TSFI used by administrators.

179 TSFI in the first category are varied in their appearance in a TOE. Most commonly interfaces are thought of as those described in terms of Application Programming Interfaces (APIs), such as kernel calls in a Unix-like operating system. However, interfaces also may be described in terms of menu choices, check boxes, and edit boxes in a GUI; parameter files (the *.INI files and the registry for Microsoft Windows systems); and network communication protocols at all levels of the protocol stack.

180 TSFI in the second category are more complex. While there are three cases that need to be considered (discussed below), for all cases there is an "additional" requirement that the functions that an administrator uses to perform their duties—as documented in administrative guidance—also are part of the TSFI and must be documented and shown to work correctly. The individual cases are as follows:

a) The administrative tool used is also accessible to untrusted users, and runs with some "privilege" itself. In this case the TSFI to be described are similar to those in the first category because the tool itself is privileged.

b) The administrative tool uses the privileges of the invoker to perform its tasks. In this case, the interfaces supporting the activities that the administrator is directed to do by the administrative guidance (AGD_ADM, including FMT_* actions) are part of the TSFI. Other interfaces supported by the tool that the administrator is directed not to use (and thus play no role in supporting the TSP), but that are accessible to non-administrators, are not part of the TSFI because there are no privileges associated with their use. Note that this case differs from the previous one in that the tool does not run with privilege, and therefore is not in and of itself interesting from a security point of view. Also note that if FPT_SEP is included in the ST, the executable image of such tools need to be protected so that an untrusted user cannot replace the tool with a "trojan" tool.

c) The administrative tool is only accessible to administrative users. In this case the TSFI are identified in the same manner as the previous case. Unlike the previous case, however, the evaluator ascertains that an untrusted user is unable to invoke the tool if FPT_SEP is included in the ST.

181 It is also important to note that some TOEs will have interfaces that one might consider part of the TSFI, but environmental factors remove them from consideration (an example is the case of interface group B1 discussed earlier). Most of these examples are for TOEs to which untrusted users have restricted access. For example, consider a firewall that untrusted users only have access to via the network interfaces, and further that the network interfaces available only support packet-passing (no remote administration, no firewall-provided services such as telnet). Further suppose that the firewall had a command-line interface that logged-in administrators could use to administer the system, or they could use a GUI-based tool that essentially translated the GUI-based checkboxes, textboxes, etc., into scripts that invoked the command-line utilities. Finally, suppose that the administrators were directed in the administrative guidance to use the GUI-based tool in administering the firewall. In this case, the command-line interface does not have to be documented because it is inaccessible to untrusted users, and because the administrators are instructed not use it.

182 The term "administrator" above is used in the sense of an entity that has complete trust with respect to all policies implemented by the TSF. There may be entities that are trusted with respect to some policies (e.g., audit) and not to others (e.g., a flow control policy). In these cases, even though the entity may be referred to as an "administrator", they need to be treated as untrusted users with respect to policies to which they have no administrative access. So, in the previous firewall example, if there was an auditor role that was allowed direct log-on to the firewall machine, the command-line interfaces not related to audit are now part of the TSFI, because they are accessible to a user that is not trusted with respect to the policies the interfaces provide access to. The point is that such interfaces need to be addressed in the same manner as previously discussed.

183 Hardware interfaces exist as well. Functions provided by the BIOS of various devices may be visible through a "wrapper" interface such as the IOCTLs in a Unix operating system. If the TOE is or includes a hardware device (e.g., a network interface card), the bus interface signals, as well as the interface seen at the network port, must be considered "interfaces." Switches that can change the behaviour of the hardware are also part of the interface.

184 As indicated above, an interface exists at the TSF boundary if it can be used (by an administrator; untrusted user; or another TOE) to affect the behaviour of the TSF. The requirements in this family apply to all types of TSFI, not just APIs.

185 All TSFI are security relevant, but some interfaces (or aspects of interfaces) are more critical and require more analysis than other interfaces. If an interface plays a role in enforcing any security policy on the system (that is, if the effects of the interface can be traced to one of the SFRs levied on the TSF), then that interface is security enforcing. Such policies are not limited to the access control policies, but also refer to any functionality provided by one of the SFRs contained in the ST (with exceptions for FPT_SEP and FPT_RVM as detailed below). Note that it is possible that an interface may have various effects and exceptions, some of which may be security enforcing and some of which may not.

186 FPT_SEP and FPT_RVM are SFRs that require a different type of analysis from other SFRs. These requirements are architecturally related, and their implementation (or lack thereof) is not easily (or efficiently) testable at the TSFI. From a terminology standpoint, although implementation (and the associated analysis) of FPT_SEP and FPT_RVM is critical to the trustworthiness of the system, these two SFRs will not be considered as SFRs that are applicable when determining the set of security-enforcing TSFIs as defined in the previous paragraph.

Editor Note: Although FPT_SEP and FPT_RVM are excepted here (and addressed in ADV_ARC), there still may be mechanisms in the TSF that are used in meeting these requirements that could and should be tested. The ATE requirements should be modified to ensure that any such testable mechanisms used to implement FPT_SEP (and possibly FPT_RVM) are included (rather than just depending upon testing of "TSFI", since TSFI are defined by the FSP requirements).

187 Interfaces (or parts of an interface) that need only to function correctly in order for the security policies of the system to be preserved are termed security supporting. A security supporting interface typically plays a role in supporting the architectural requirements (FPT_SEP or FPT_RVM), meaning that as long as it can be shown that it does not allow the TSF to be compromised or bypassed no further analysis against SFRs is required. In order for an interface to be security supporting it must have no security enforcing aspects. In contrast, a security enforcing interface may have security supporting aspects (for example, the ability to set the system clock may be a security enforcing aspect of an interface, but if that same interface is used to display the system date that effect may only be security supporting).

188 A key aspect for the assurance associated with this component is the concept of the evaluator being able to verify that the developer has correctly categorized the security enforcing and security supporting interfaces. As more information about the TSFI becomes available, the greater the assurance that can be gained that the interfaces are correctly categorized/analyzed. The requirements are structured such that, at the lowest level, the information required for security supporting interfaces is the minimum necessary in order for the evaluator to make this determination in an effective manner. At higher levels, more information becomes available so that the evaluator has greater confidence in the designation.

189 For the purposes of the requirements, interfaces are specified (in varying degrees of detail) in terms of their parameters, parameter descriptions, effects, exceptions, and error messages. Additionally, the purpose of each interface, and the way in which the interface is used (both from the point of view of the external stimulus (e.g., the programmer calling the API, the administrator changing a setting in the registry) and the effect on the TSFI that stimulus has) must be specified. This description of method of use must also specify how those administrative interfaces that are unable to be successfully invoked by untrusted users (case "c" mentioned above) are protected.

190 Parameters are explicit inputs to and outputs from an interface that control the behaviour of that interface. For examples, parameters are the arguments supplied to an API; the various fields in a packet for a given network protocol; the individual key values in the Windows Registry; the signals across a set of pins on a chip; etc.

191 A parameter description tells what the parameter is in some meaningful way. For instance, the interface "foo(i)" could be described as having "parameter i which is an integer"; this is not an acceptable parameter description. A description such as "parameter i is an integer that indicates the number of users currently logged in to the system." is required.

192 Effects of an interface describe what the interface does. The effects that need to be described in an FSP are those that are visible at any external interface (for instance, audit activity caused by the invocation of an interface (assuming audit requirements are included in the ST) should be described, even though that "effect" is generally not visible through the invoked interface), not necessarily limited to the one being specified. The "effect" of an API call is not just the error code it returns, but the state changes that occur in the TSF as a result of that call. Also, depending on the parameters of an interface, there may be many different effects (for instance, an API might have the first parameter be a "subcommand", and the following parameters be specific to that subcommand. The IOCTL API in some Unix systems is an example of such an interface).

193 Exceptions refer to the processing associated with "special checks" that may be performed by an interface. An example would be an interface that has a certain set of effects for all users except the Superuser; this would be an exception to the normal effect of the interface. Use of a privilege for some kind of special effect would also be covered in this topic.

194 Documenting the errors associated with the TSF is not as straight-forward as it might appear, and deserves some discussion. A general principle is that errors generated by the TSF that are visible to the user should be documented. These errors can be the direct result of invoking a TSFI (an API call that returns an error); an indirect error that is easily tied to a TSFI (setting a parameter in a configuration that is error-checked when read, returning an immediate notification); or an indirect error that is not easily tied to a TSFI (setting a parameter that, in combination with certain system states, generates an error condition that occurs at a later time. An example might be resource exhaustion of a TSF resource due to setting a parameter to too low of a value).

195 Errors can take many forms, depending on the interface being described. For an API, the interface itself may return an error code; set a global error condition, or set a certain parameter with an error code. For a configuration file, an incorrectly configured parameter may cause an error message to be written to a log file. For a hardware PCI card, an error condition may raise a signal on the bus, or trigger an exception condition to the CPU.

196 For the purposes of the requirements, errors are divided into two categories. The first category includes direct errors, which are directly related to a TSFI; examples are API calls and parameter-checking for configuration files. For this category of errors, the functional specification must document all of the errors that can be returned as a result of invoking the interface, and, from it, a reader should be able to associate an interface with the errors it is capable of generating. The second category includes indirect errors, which are errors that are not directly tied to the invocation of a TSFI, but which are reported to the user as a result of processing that occurs in the TSF. It should be noted that the condition that causes the indirect error can be documented; it is generally much harder to document all the ways in which that condition can occur.10 Because of the difficulty associated with documenting all of the ways to cause an error, and because of the cost of documenting all indirect errors compared to the benefit of having them documented, indirect errors are not required to be fully documented until the highest EALs.

197 If an interface plays a role in enforcing any security policy on the system, that interface is security enforcing. Such policies are not limited to the access control policies, but also refer to any functionality provided by one of the SFRs contained in the ST.

198 An important factor in determining the set of security-supporting interfaces is whether there exists the threat of accidental or malicious bypass or misuse of the TSF (generally indicated by including components from the FPT_RVM and/or FPT_SEP families in the ST). In the case that there is such a threat, the TSFI are can be categorized as security enforcing and security supporting as discussed above. In the case where there isn't this threat, then there are only security enforcing interfaces, and these are defined with respect to the security functions that the TSF implements. It should be noted that there is no such thing as a "non-security-relevant TSFI". An interface either allows access to a portion of the TSF or it doesn't. If it allows access to the TSF, then it is by definition security relevant, and is either security enforcing or security supporting.

199 Having defined the terms, the following amplification on component levelling can be given. At all levels, the purpose and method of use of the TSFI must be provided.

200 At ADV_FSP.1, because there is no dependency on the HLD requirements (which require that the TSFI be identified in the high-level design), there is lower assurance that the functional specification of the TSFI is complete. At this lowest level, the developer is required provide the purpose and method of use for the TSFI. They need to identify parameters, and to provide parameter descriptions for those parameters, for all TSFI.

201 At ADV_FSP.2, there is a dependency on ADV_HLD which requires that the TSF be identified (which is true for all higher ADV_FSP components). Now more assurance can be gained that the functional specification covers the entire TSF. Additionally, for the TSFI designated as security enforcing, they have to describe the security-enforcing effects, security-enforcing exceptions, and direct error messages that can occur in relation to security-enforcing behavior associated with the TSFI. At this level, the developer performs the initial determination of the TSF (and thus the TSFI) and designates the interfaces as security enforcing or security supporting, which

[10] *This may even be impossible, if the error message is for a condition that the programmer does not expect to occur, but is inserted as part of "defensive programming."*

the evaluators confirm through their evaluation activities. The additional information provided by the developer allows the evaluator to have some assurance that the developer has correctly designated the interfaces.

202 At ADV_FSP.3, the developer must now, in addition to the information required at ADV_FSP.2 for the effects, exceptions, and error messages, provide enough information about the security supporting effects, exceptions, and error messages associated with the TSFI so that it can be determined by the evaluator that they are not security enforcing. At this level, the evaluator has more information to confirm the developer's determination of security enforcing vs. security supporting.

203 At ADV_FSP.4, the developer needs to provide complete documentation of all TSFI. This means that in addition to the documentation required for the TSFI at ADV_FSP.3, the developer also needs to provide a description of all exceptions, error messages (direct and indirect), and effects for all TSFI. This provides the evaluators with information that, when analysed, will increase the assurance that all of the security enforcing aspects of the TSFI have been characterized, thus leading to more complete testing and increased confidence in the vulnerability assessment activities.

204 At ADV_FSP.5 and ADV_FSP.6, the only significant additional requirements are the presentation of the information needs to be semi-formal and formal, respectively. At ADV_FSP.6 slightly more information is required for indirect error messages as well.

205 The ADV_FSP.*.2E elements within this family define a requirement that the evaluator determine that the functional specification is an accurate and complete instantiation of the TOE security functional requirements. This provides a direct correspondence between the TOE security functional requirements and the functional specification, in addition to the pairwise correspondences required by the ADV_RCR family. Although the evaluator may use the evidence provided in ADV_RCR as an input to making this determination, ADV_RCR cannot be the basis for a positive finding in this area. The requirement for completeness is intended to be relative to the level of abstraction of the functional specification.

206 It should be recognized that there exist functional requirements whose functionality is manifested wholly or in part architecturally, rather than through a specific mechanism. An example of this is FPT_SEP, where one cannot specify the "set" of interfaces that implement the self-protection "function." This type of "functionality" typically is verified by examination of the design and (at the higher assurance levels) implementation of the TSF. Another example is the implementation of mechanisms implementing the FPT_RIP requirements. Such mechanisms typically are implemented to ensure an effect isn't present, which is difficult to test and typically has the be verified through analysis as well.

207 In the cases where such functional requirements are included in the ST, it is expected that evaluators recognize that there may be SFRs of this type that have no interfaces, and that this should not be considered a deficiency in the functional specification. The terminology in the ADV_FSP.*.2E requirements referring to "user-visible TOE security functional requirements" is used to exclude this type of analysis for this family; the analysis for these types of requirements is required by the ADV_ARC family components.

208 In the context of the level of formality of the functional specification, informal, semiformal and

formal are considered to be hierarchical in nature. Thus, ADV_FSP.[1-4] may also be met with either a semiformal or formal functional specification, provided that it is supported by informal, explanatory text where appropriate. In addition, ADV_FSP.5 may also be met with a formal functional specification.

Dependencies for ADV_FSP_EXP.6

ADV_HLD_EXP.1 Descriptive High-level Design

ADV_RCR_EXP.1 Subsystem Correspondence Demonstration

E.5 Rationale for ADV_HLD_EXP

High-level design (ADV_HLD)

Objectives

209 The high-level design of a TOE provides both context for a description of the TSF, and a thorough description of the TSF in terms of major structural units (i.e. subsystems). It relates these units to the functions that they provide. The high-level design requirements are intended to provide assurance that the TOE provides an architecture appropriate to implement the TOE security functional requirements.

210 To provide context for the description of the TSF, the high-level design describes the entire TOE at a high level in terms of subsystems. From this description the reader should be able to distinguish between the subsystems that are part of the TSF and those that are not. The remainder of the high-level design document then describes the TSF in more detail.

211 The high-level design provides a further-refined description of the TSF from that presented in the functional specification. The functional specification provides a description of what the TSF does at its interface; the high-level design provides more insight into the TSF by describing how the TSF works in order to perform the functions specified at the TSFI. For each subsystem of the TSF, the high-level design identifies the TSFI implemented in the subsystem, describes the purpose of the subsystem and how the implementation of the TSFI (or portions of the TSFI) is designed. The interrelationships of subsystems are also defined in the high-level design. These interrelationships will be represented as data flows, control flows, etc. among the subsystems. It should be noted that this description is at a high-level; low-level implementation detail is not necessary at this level of abstraction.

Component levelling

212 The components in this family are levelled on the basis of the amount of information that is required to be presented about the subsystems of the TSF, and on the degree of formalism

required of the high-level design.

Application note

213 The developer is expected to describe the design of the TSF in terms of subsystems. The term "subsystem" is used here to express the idea of decomposing the TSF into a relatively small number of parts. While the developer is not required to actually have "subsystems", the developer is expected to represent a similar level of decomposition. For example, a design may be similarly decomposed using "layers", "domains", or "servers".

214 A security-enforcing subsystem is a subsystem that provides mechanisms for enforcing an element of the TSP, or directly supports a subsystem that is responsible for enforcing the TSP. If a subsystem provides (implements) a security enforcing TSFI, then the subsystem is security enforcing. If a subsystem does not provide any security enforcing TSFIs, its mechanisms still must preserve the security of the TSF; such subsystems are termed security supporting.

215 The set of SFRs that determine the TSP for the purposes of ADV_HLD components do not include FPT_SEP and FPT_RVM. Those two architectural functional requirements require a different type of analysis than that needed for all other SFRs. A security-enforcing subsystem is one that is designed to implement an SFR other than FPT_SEP and FPT_RVM; requirements pertaining to design information and justification for the FPT_SEP and FPT_RVM requirements are contained in the ADV_ARC components.

216 As more information is described pertaining to a subsystem, increasing assurance can be gained that 1) the design of the subsystem supports the security functions; 2) the design of the subsystem preserves the security of the system; and 3) the subsystem is correctly characterized as security enforcing or security supporting.

217 For TOEs that contain requirements on the IT Environment, there is a requirement to supply information at the "subsystem" level of detail that describes the functionality that the TOE is dependant upon. This description will not be extremely detailed, as the actual design and implementation of the dependant functionality is likely not known. However, the description should be detailed enough so that the evaluator can determine how the TOE/TSF interacts with the functionality provided by the IT Environment, and use this information to help determine whether the interfaces (ADV_FSP, ADV_CMP) are completely described.

218 Given this discussion, a summary of the different components of this family can now be given.

219 For ADV_HLD.1, the developer describes the TOE in terms of subsystems, identifying those that are part of the TSF (both security enforcing and security supporting) and those that are not. This description also must provide enough detail so that the reader can understand why the subsystems that are not part of the TSF are so designated. In addition, the developer provides a somewhat detailed description of the security-enforcing subsystems, and further describes the security-enforcing functionality associated with those subsystems. Additionally, the developer identifies and describes the interactions between the security-enforcing subsystems.

220 At ADV_HLD.2, the developer must now provide some description of security-supporting functionality. In order to provide confidence that the subsystems have been classified correctly, and to provide a greater understanding of the system so that this confidence is increased, they

must summarize functionality not described for security-enforcing subsystems at ADV_HLD.1; the functionality of the non-security-enforcing modules; and the interactions between subsystems that were not described in ADV_HLD.1. They do not, however, need to provide the same level of detail for the security-supporting subsystems as they do for the security-enforcing subsystems.

221 At ADV_HLD.3, a complete description of all of the subsystems of the TSF, and their interactions, is needed. This provides more assurance that the description of the security-enforcing aspects of the subsystems is complete.

222 ADV_HLD.4 adds rigor (semiformal specification) to the descriptions of the TSF subsystems and their interactions. It is important to note that the formality only applies to the description of the TSF; describing the TOE in terms of subsystems in order to identify the subsystems that comprise the TSF can still be presented in an informal manner if the developer desires.

223 As with the ADV_FSP components, the set of SFRs that determine the TSP for the purposes of this component do not include FPT_SEP and FPT_RVM. Those two architectural functional requirements require a different type of analysis than that needed for all other SFRs. A security-enforcing subsystem is one that is designed to implement an SFR other than FPT_SEP and FPT_RVM; the design information and justification for the FPT_SEP and FPT_RVM requirements is given as a result of the ADV_ARC components.

224 The ADV_HLD component requires that the developer must identify all subsystems of the TSF (not just the security-enforcing ones). For the lower components, the security-enforcing aspects of the subsystems need to be described in more detail than the security-supporting aspects. The descriptions for the security-enforcing aspects should provide the reader with enough information to determine how the implementation of the SFRs is designed, while the description for the security-supporting aspects should provide the reader enough assurance to determine that 1) all security-enforcing behavior has been identified and 2) the subsystems or portions of subsystems that are security supporting have been correctly classified.

225 The ADV_HLD.*.2E elements within this family define a requirement that the evaluator determine that the high-level design is an accurate and complete instantiation of the TOE security functional requirements. This provides a direct correspondence between the TOE security functional requirements and the high-level design, in addition to the pairwise correspondences required by the ADV_RCR family. Although the evaluator may use the evidence provided in ADV_RCR as an input to making this determination, ADV_RCR cannot be the basis for a positive finding in this area. The requirement for completeness is intended to be relative to the level of abstraction of the high-level design.

226 Note the term “user visible” is used to highlight the fact that FPT_SEP and FPT_RVM are not explicitly analysed in this fashion by the ADV_HLD.*.2E elements; the analysis for those requirements is done as part of the activity for the ADV_ARC components.

227 In the context of the level of formality of the high-level design, informal and semiformal are considered to be hierarchical in nature. Thus, ADV_HLD.1.3C, ADV_HLD.2.3C, and ADV_HLD.3.3C may also be met with a semiformal high-level design.

Dependencies for ADV_HLD_EXP.4

ADV_FSP_EXP.1 Descriptive Functional Specification

ADV_RCR_EXP.1 Subsystem Correspondence Demonstration

E.6 Rationale for ADV_IFA_EXP

Information Availability (ADV_IFA)

Objectives

228 This family and the ADV_CMP family are intended to be used to provide information to those persons responsible for integrating, or assessing the integration of, the TOE with other IT components in manner that provides confidence that the composed system preserves the security functions of the TOE. The issue of integrating an evaluated component with other components is commonly called the composition problem. There are two distinct aspects to this issue: composition "in the large" and composition "in the small". While there may be some subjectivity in categorizing the composition of specific sets of TOEs, in general composition in the large refers to composing relatively complete products or systems (an OS, firewall, or secure router) into a larger system. In contrast, composition in the small refers to integrating a TOE that has requirements on the IT environment into that environment. Integrating an evaluated database onto an OS platform and integrating an OS onto a hardware platform are both example of composition in the small.

229 This family and the ADV_CMP family are targeted at the problem of composition "in the small." The are two aspects to this problem; those for the "base TOE" (covered by this family) and those for the "dependant TOE" (covered by the ADV_CMP family). A base TOE is a TOE that may or may not have requirements on the IT Environment, but is expected to have other TOEs combined with it in an operational environment. An operating system would be an example of such a TOE. A dependant TOE is a TOE that has requirements on the IT Environment, which may be satisfied by one or more base TOEs. A database or firewall application would be an example of a dependant TOE.

230 There are no requirements or actions specified that an integrator would follow to actually compose the various components, nor evaluate the result of that composition effort. Rather, the purpose of this family is to ensure that the information to perform an integration activity is provided by the TOE developer to system integrators, persons wishing to analyze a composed system, or other parties that need that information.

Component levelling

231 There is only one component in this family.

Application notes

232 Throughout this family the term “system integrator” is used to refer to the end users of the information produced as a result of meeting the requirements of this family. It should be noted that system integrators are not the only consumers of this information. Members of teams responsible for assessing the overall security of a system in which components have been composed would also find this information key to providing them with a measure of assurance that the security mechanisms have not be compromised (or the extent to which the mechanisms may have been compromised) as a result of the integration activity.

233 The crux of the composition issue is the desire to evaluate a component once, then use that component in a variety of systems without requiring further evaluation of the entire system, while maintaining a base level of trust in the overall (integrated) system. In practice, this desire does not seem achievable in the general sense for several reasons:

a) The evaluated version of the component does not include functionality that is used in the composed system, introducing unknown interactions between that “new” functionality and the evaluated component;

b) The interactions between the evaluated component and other components in the system are not documented, which may leave undiscovered interfaces that can be used to compromise the integrity of the composed system; and

c) A component in a system may be privileged with respect to security policies enforced by other components in the system, thus creating a potential for the privileged component to compromise the integrity of the other components in the system.

234 One approach to alleviating the composition problem is to in a sense remove “composition” from consideration. In this approach, one would specify the entire end system in terms of an overall architecture and individual components, providing enough detail so that the end system would meet its functional goals when the components were assembled, and the overall security policy of the system would be preserved. There are two issues with this approach: first, it does not allow for arbitrary composition of evaluated products; and second, it requires extremely detailed specifications in order to ensure that the security properties are preserved once the components are “composed.”

235 Although the composition problem cannot otherwise be fully addressed, information pertaining to evaluated products can be made available that mitigate the risks in composing systems and shorten the assessment time necessary to determine that the level of assurance provided by the composed system is adequate. Using this approach, this family specifies documentation requirements on the developer of the TOE that will mandate such documentation be produced and be made available to integrators. Although there will still be issues when composing evaluated TOEs with non-evaluated TOEs, composing evaluated TOEs in a trustworthy fashion should be much easier.

236 For the component in this family, the developer must make available its functional specification (that is, the evidence supplied to meet the ADV_FSP requirement), test coverage analysis

(supplied for ATE_COV and ATE_DPT), and test procedure descriptions (provided for the ATE_FUN component). They also must supply composition information (supplied for ADV_CMP). It should be noted that for TOEs that make no calls out to the IT environment there will be no composition information generated, so only the functional specification and test information will need to be supplied in such cases.

237 While this information is required to be made available to systems integrators, it is also important to ensure that the developer is not placed at a competitive disadvantage by being forced to supply a competitor with sensitive information as a result of meeting these requirements. The intent of the requirements in this component is not for the developer to make the information publicly available; rather, the developer must agree to make it available under conditions set forth in the Integrators Disclosure Agreement (ADV_CMP.1.3D) which could include, for instance, a non-competition agreement or a non-disclosure agreement. Making this information available allows the integrator to use information generated for the dependant TOE (the ADV_CMP components) to identify interfaces used in the base TOE, and then determine whether those interfaces were considered as part of the TSFI, and the extent to which the interface was tested in the base product.

Dependencies for ADV_IFA_EXP.1

ADV_CMP_EXP.1 Basic Composition Information

ADV_FSP_EXP.1 Descriptive Functional Specification

ADV_HLD_EXP.1 Descriptive High-Level Design

ATE_COV.1 Evidence of Coverage

ATE_FUN.1 Functional Testing

E.7 Rationale for ADV_IMP_EXP

Implementation Representation (ADV_IMP)

Objectives

238 The function of the ADV_IMP family is for the developer to provide the actual implementation of the TOE in a form that this understandable to the evaluators. This is provided for use in analysis activities (analyzing the low-level design, for instance) to demonstrate that the TOE conforms its design and to provide a basis for analysis in other areas of the evaluation (e.g. the search for vulnerabilities). The implementation representation is expected to be in a form that captures the detailed internal workings of the TSF. This may be software source code, firmware source code, hardware diagrams and/or chip specifications, etc. At higher levels, proof must also be offered that portions of the implementation representation match the implementation itself

(e.g., object code).

Component levelling

239 The components in this family are levelled on the basis of the amount of information provided in addition to the implementation representation, and the activities performed on this additional information.

Editor Note: ***As currently written, a distinction is made at the .3 level between the portion of the TSF implemented in hardware and that implemented in compiled or interpreted entities (software and firmware). The .3 level requires that the implementation essentially be "reverse-engineered", which is easier to do for software/firmware than it is for hardware. For software/firmware only significant expertise is required. For hardware, not only is significant expertise required, but expensive equipment is also required to physically examine, measure, and manipulate the hardware. Because hardware is generally functionally tested in a more thorough manner, and because the added assurance from reverse-engineering the hardware does not initially appear to be commensurate with the costs involved in obtaining that assurance, the additional requirements at the .3 level will only apply to the software/firmware portions of the TSF implementation.***

Application note

240 The implementation representation is used to express the notion of the least abstract representation of the TSF, specifically the one that is used to create the TSF itself without further design refinement. Source code that is then compiled or a hardware diagrams and/or chip specifications that are used to build the actual hardware are examples of parts of an implementation representation.

241 The entire implementation representation is provided to ensure that analysis activities are not curtailed due to lack of information. The implementation representation is supplied to allow analysis of other TOE design decompositions (e.g., functional specification, low-level design), and to gain confidence that the higher-level security mechanisms described in the design actually appear to be implemented in the TOE. Conventions in some forms of the implementation representation may make it difficult or impossible to determine from just the implementation representation itself what the actual result of the compilation or run-time interpretation will be. For example, compiler directives for C language compilers will cause the compiler to exclude or include entire portions of the code. For this reason, it's important that such "extra" information be provided so that the implementation representation can be accurately determined.

242 Even given this most detailed description of the TSF, there is still a question of whether the actual implementation reflects what is written in the implementation representation. It is possible that compiler functions (such as optimization) or outright errors (either due to maliciousness or programming errors) could cause the implementation that runs on the hardware base to be different from the implementation representation in such a way that the security mechanisms of the system are compromised. This compromise will likely be one involving protection of the

TSF itself, since functional compromises (unless malicious) should be detectable through testing.

243 There are two methods defined for addressing this threat. The first is to use tools designed for debugging the TSF to step through the implementation to ensure that the requirements are implemented accurately, and that the functionality is security preserving. While this adds assurance over examination of the implementation and functional testing (because observations are being made internal to the TSF, rather than just at the external interface), it is inadequate for detecting artifacts that may cause the policy enforcement mechanisms to fail (e.g., buffer overflows resulting from compiler optimizations, and therefore not visible from review of the implementation representation).

244 The second, more definitive, method is to start with the implementation itself (the compiled code that runs on the hardware) and de-compile it. This de-compiled version can then be compared to functionality described in other evidence supplied for the evaluation (e.g., low-level design, the implementation representation) to determine that it is security preserving. Comparing it to the implementation representation alone may be problematic, since optimizations may affect ordering of statements, storage of variables, etc.

245 What is likely required for this method is a combination of reverse-compilation and run-time activities, calling for the developer to supply tools used in the debugging the TSF implementation. Stepping through the code and being able to affect the implementation constructs will allow confidence that the implementation completely and accurately reflects its representation. This adds assurance over the first method, since it is unlikely that all functionality can be exercised without some knowledge of the de-compiled implementation.

246 The analog of this activity for hardware implementations is much more difficult to express, since the process of creating the hardware implementation is, unlike compilation, not as easily reversible, and there is generally not a notion of "stepping through hardware", although some provision may have been made through the installation of test pins and special logic. Such analysis is possible using specialized procedures and test equipment, but the cost (in terms of time, expertise required, and expense of obtaining the equipment) of performing the analysis outweighs the potential gains from such an analysis at this time.

247 Some forms of the implementation representation may require additional information because they introduce significant barriers to understanding and analysis. Examples include "shrouded" source code or source code that has been obfuscated in other ways such that it prevents understanding and/or analysis. These forms of implementation representation typically result from by taking a version of the implementation representation that is used by the TOE developers and running a shrouding or obfuscation program on it. While the shrouded representation is what is compiled and may be "closer" to the implementation (in terms of structure) than the original, un-shrouded representation, supplying such a version may cause significantly more time to be spent in analysis tasks involving the representation. When such forms of representation are created, the components require details on the shrouding tools/algorithms used so that the un-shrouded representation can be supplied, and the additional information can be used to gain confidence that the shrouding process does not compromise any security mechanisms.

248 The implementation representation is transformed into the actual implementation. For hardware, this may involve creating integrated circuits from schematics or hardware description languages.

For software, this may involve creating object code modules from source code files. It may even be the source code itself in the case where the language is interpreted. An external representation of the implementation is the term used to describe discrete units of the implementation are packaged once they are transformed from the implementation representation.

249 For software this is the entity (for example, a file) that is output by the compiler that executes directly on the hardware, or the entity that is input by the interpreter. The key concept is that the external representation of the implementation is the actual implementation that will perform the functions of the TSF.

250 The components in this family are levelled based on the amount of information relating to the implementation that is supplied, and the confidence that is obtained that the actual implementation reflects the security design of the TSF. The ADV_IMP.1 component requires the complete implementation representation to be supplied, and any additional information needed to accurately interpret the implementation representation. At ADV_IMP.2, tools must be supplied and used to provide assurance that at the implementation level, the security mechanisms appear to be security preserving. At ADV_IMP.3, information and tools must be provided so that analysis can be performed verifying that the software implementation accurately reflect the SFRs.

Dependencies for ADV_IMP_EXP.3

ADV_LLD_EXP.3 Complete Low-Level Design

ADV_RCR_EXP.1 Subsystem Correspondence Demonstration

ALC_TAT.1 Well-defined Development Tools

E.8 Rationale for ADV_INT_EXP

TSF Internals (ADV_INT)

Objectives

251 This family addresses the internal structure of the software TSF. Requirements are presented for modular decomposition, layering, principle of least privilege, and minimisation of the amount of non-TSP-enforcing functionality within the TSF. When used effectively, modularity and layering provide many assurance benefits including data hiding and abstraction. These requirements, when applied to the internal structure of the TSF, should result in improvements that aid both the developer and the evaluator in understanding the TSF, and also provides the basis for designing and evaluating test suites. Further, improving understandability of the TSF should assist the developer in simplifying its maintainability. The principal goal achieved by inclusion of the requirements from this class in a PP/ST is understandability of the TSF.

252 Modular design aids in achieving understandability by clarifying what dependencies a module has on other modules (coupling), by including in a module only tasks that are strongly related to each other (cohesion), and by illuminating the design of a module by using internal structuring and reduced complexity. The use of modular design reduces the interdependence between elements of the TSF and thus reduces the risk that a change or error in one module will have effects throughout the TOE. Its use enhances clarity of design and provides for increased assurance that unexpected effects do not occur. Additional desirable properties of modular decomposition are a reduction in the amount of redundant or unneeded code.

253 Another key component to reducing complexity is the use of coding standards. Coding standards are used as a reference to ensure programmers generate code that can be easily understood by individuals (e.g., code maintainers, code reviewers, evaluators) that are not intimately familiar with the nuances of the functions performed by the code. For example, coding standards ensure that meaningful names are given to variables and data structures, the code has a structure that is similar to code developed by other programmers, loops used in the code are understandable (e.g., leaving a loop to another section of code and returning is undesirable), the use of pointers to variables/data structures is straightforward, and the code is suitably commented (inline and/or by a preamble). The use of coding standards helps to eliminate errors in code development and maintenance, and assists the development team in performing code walk-throughs. Some aspects of coding standards are specific to a given program language (e.g., the C language may have a different standard from the Java language or assembly level code). It is expected that the coding standards be appropriately followed for the employed programming language(s). The requirements in this component allow for exceptions to the adherence of coding standards that may be necessary for reasons of performance, or some other factors, but these deviations must be justified (on a per module basis) as to why they are necessary. Any justification provided must address why the deviation does not unduly introduce complexity into the module, since ultimately, the goal of adhering to coding standards is to improve clarity.

254 The use of layering to separate levels of abstraction and minimise circular dependencies further enables a better understanding of the TSF, providing more assurance that the TOE security functional requirements are accurately and completely instantiated in the implementation.

255 Minimising the amount of functionality in the TSF allows the evaluator as well as the developer to focus only on that functionality which is necessary for TSP enforcement, contributing further to understandability and further lowering the likelihood of design or implementation errors.

256 The incorporation of modular decomposition, layering and minimization into the design and implementation process must be accompanied by sound software engineering considerations. A practical, useful software system will usually entail some undesirable coupling among modules, some modules that include loosely-related functions, and some subtlety or complexity in a module's design. These deviations from the ideals of modular decomposition are often deemed necessary to achieve some goal or constraint, be it related to performance, compatibility, future planned functionality, or some other factors, and may be acceptable, based on the developer's justification for them. In applying the requirements of this class, due consideration must be given to sound software engineering principles; however, the overall objective of achieving understandability must be achieved.

257 Design complexity minimisation is a key characteristic of a reference validation mechanism, the

purpose of which is to arrive at a TSF that is easily understood so that it can be completely analysed. (There are other important characteristics of a reference validation mechanism, such as TSF self-protection and TSP non-bypassability; these other characteristics are covered by requirements from other classes.)

Component levelling

258 The components in this family are levelled on the basis of the amount of structure and minimisation required. Partial modular decomposition at ADV_INT.1 places requirements for modular decomposition on only selected parts of the TSF. At the next level, the requirements for modular decomposition are placed on the entire TSF, and the requirements for dealing with duplicate or unused code are strengthened. Layering and minimization are then introduced in the next two higher components.

Application notes

259 Several of the elements within the components for this family refer to the software architectural description. The software architectural description is at a similar level of abstraction as the low-level design, in that it is concerned with the modules of the TSF. Whereas the low-level design describes the design of the modules of the TSF, the purpose of the software architectural description is to provide evidence of modular decomposition, layering, and minimisation of complexity of the TSF, as applicable. Both the low-level design and the implementation representation are required to be in compliance with the software architectural description, to provide assurance that these TSF representations possess the required modular decomposition, layering, and minimisation of complexity.

260 The modules identified in the software architectural description are the same as the modules identified in the low-level design.

Terms, definitions and background

261 The following terms are used in the requirements for software internal structuring. Some of these are derived from the Institute of Electrical and Electronics Engineers Glossary of software engineering terminology, IEEE Std 610.12-1990.

262 *modular* decomposition: the process of breaking a system into components to facilitate design and development.

263 *cohesion* (also called *module strength*): the manner and degree to which the tasks performed by a single software module are related to one another; types of cohesion include coincidental, communicational, functional, logical, sequential, and temporal. These types of cohesion are characterised below, listed in the order of decreasing desirability.

264 *functional cohesion*: a module with this characteristic performs activities related to a single purpose. A functionally cohesive module transforms a single type of input into a single type of output, such as a stack manager or a queue manager.

265 *sequential cohesion:* a module with this characteristic contains functions each of whose output is

input for the following function in the module. An example of a sequentially cohesive module is one that contains the functions to write audit records and to maintain a running count of the accumulated number of audit violations of a specified type.

266 *communicational cohesion:* a module with this characteristic contains functions that produce output for, or use output from, other functions within the module. An example of a communicationally cohesive module is an access check module that includes mandatory, discretionary, and capability checks.

267 *temporal cohesion:* a module with this characteristic contains functions that need to be executed at about the same time. Examples of temporally cohesive modules include initialization, recovery, and shutdown modules.

268 *logical (or procedural) cohesion:* a module with this characteristic performs similar activities on different data structures. A module exhibits logical cohesion if its functions perform related, but different, operations on different inputs.

269 *coincidental cohesion:* a module with this characteristic performs unrelated, or loosely related activities.

270 *coupling:* the manner and degree of interdependence between software modules; types of coupling include call, common and content coupling. These types of coupling are characterised below, listed in the order of decreasing desirability.

271 *call:* two modules are call coupled if they communicate strictly through the use of their documented function calls; examples of call coupling are data, stamp, and control, which are defined below.

- *data:* two modules are data coupled if they communicate strictly through the use of call parameters that represent single data items.
- *stamp:* two modules are stamp coupled if they communicate through the use of call parameters that comprise multiple fields or that have meaningful internal structures.
- *control:* two modules are control coupled if one passes information that is intended to influence the internal logic of the other.

272 *common:* two modules are common coupled if they share a common data area or a common system resource. Global variables indicate that modules using those global variables are common coupled11.

273 Common coupling through global variables is generally allowed, but only to a limited degree. For example, variables that are placed into a global area, but are used by only a single module, are inappropriately placed, and should be removed. Other factors that need to be considered in assessing the suitability of global variables are:

The number of modules that modify a global variable: In general, only a single module should be allocated the responsibility for controlling the contents of a global variable, but

[11] *It can be argued that modules sharing definitions, such as data structure definitions, are common coupled. However, for the purposes of this analysis, shared definitions are considered acceptable, but are subject to the cohesion analysis.*

there may be situations in which a second module may share that responsibility; in such a case, sufficient justification must be provided. It is unacceptable for this responsibility to be shared by more than two modules. (In making this assessment, care should be given to determining the module actually responsible for the contents of the variable; for example, if a single routine is used to modify the variable, but that routine simply performs the modification requested by its caller, it is the calling module that is responsible, and there may be more than one such module). Further, as part of the complexity determination, if two modules are responsible for the contents of a global variable, there should be clear indications of how the modifications are coordinated between them.

The number of modules that reference a global variable: Although there is generally no limit on the number of modules that reference a global variable, cases in which many modules make such a reference should be examined for validity and necessity.

274 *content*: two modules are content coupled if one can make direct reference to the internals of the other (e.g. modifying code of, or referencing labels internal to, the other module). The result is that some or all of the content of one module are effectively included in the other. Content coupling can be thought of as using unadvertised module interfaces; this is in contrast to call coupling, which uses only advertised module interfaces.

275 *call tree*: a diagram that identifies the modules in a system and shows which modules call one another. All the modules named in a call tree that originates with (i.e., is rooted by) a specific module are the modules that directly or indirectly implement the functions of the originating module.

276 *software engineering:* the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software. As with engineering practices in general, some amount of judgement must be used in applying engineering principles. Many factors affect choices, not just the application of measures of modular decomposition, layering, and minimisation. For example, a developer may design a system with future applications in mind that will not be implemented initially. The developer may choose to include some logic to handle these future applications without fully implementing them; further, the developer may include some calls to as-yet unimplemented modules, leaving call stubs. The developer's justification for such deviations from well-structured programs will have to be assessed using judgement, as well as the application of good software engineering discipline.

277 *complexity:* this is a measure of how difficult software is to understand, and thus to analyse, test, and maintain. Reducing complexity is the ultimate goal for using modular decomposition, layering and minimization. Controlling coupling and cohesion contributes significantly to this goal.

278 A good deal of effort in the software engineering field has been expended in attempting to develop metrics to measure the complexity of source code. Most of these metrics use easily computed properties of the source code, such as the number of operators and operands, the complexity of the control flow graph (cyclomatic complexity), the number of lines of source code, the ratio of comments to executable code, and similar measures. Coding standards have been found to be a useful tool in generating code that is more readily understood.

279 This family calls for the evaluator to perform a complexity analysis in all components. The developer may be required to support the evaluation team's activities in this area by providing a form of the implementation representation that allows complexity analysis tools to be used to measure some of the properties of the source code.

280 *layering:* the design software such that separate groups of modules (the layers) are hierarchically organized to have separate responsibilities such that one layer depends only on layers below it in the hierarchy for services, and provides its services only to the layers above it. Strict layering adds the constraint that each layer receives services only from the layer immediately beneath it, and provides services only to the layer immediately above it.

Mininization of Complexity (ADV_INT_EXP.4)

Objectives

281 This component specifies requirements that are intended to ensure the entire TSF is structured and implemented in a way that facilities an ease of understanding. A conformant TSF will be simple enough to analyse such that it can be fully understood, with the accompanying design information, by the evaluation team.

Application notes

Editor Note: Some of this introductory matter might be more appropriate in the methodology.

282 This component requires that the reference monitor property "simple enough to be analysed" is fully addressed. When this component is combined with the functional requirements FPT_RVM.1 and FPT_SEP.3, the reference monitor concept would be fully realised.

283 The component requires the use of layering in the architecture to aide in understanding of the TSF. The layers of the design are constructed such that the lowest layer in the design provides services for the next layer in the hierarchy. While exceptions are allowed for a lower layer in the hierarchy to initiate a call to a higher layer in the hierarchy, they are discouraged and require a strong justification that these types of interactions do not result in circular dependencies or pose potential recursion issues. The primary purpose of a layer architecture is that each layer can be analysed and understood without knowing the workings of the adjacent layers. This allows the TSF architecture to be decomposed into more manageable pieces for analysis by the evaluation team and the developer's staff. Strict layering is not required to satisfy this component.

284 This component addresses the minimization of complexity not only through structure, but also by the elimination of code that is not contributing to the enforcement of the TOE's defined security policies. This requires the removal of unused or "dead-code", redundant code, and code (e.g., functions, modules, macros) that may be reachable but does not play a role a role in TSP enforcement, from the final implementation (e.g., binary executable). For example, code that

exists in the implementation representation, but based upon compiler options does not get compiled into the executable would not have to be removed (e.g., debug code, hardware platform specific code); however, information must be provided that makes it clear what code in the implementation representation will exist in the resulting implementation. Exceptions of unused and redundant code existence are allowed as long as there is a suitable justification. A suitable justification for unused code may be that future product functionality has begun to be staged into the development, but is not complete. Redundant code is more difficult and is subjective. Functions that are identical would require a strong argument for their inclusion. Functions that perform a similar purpose, but differ slightly in their implementation require engineering judgement to determine the legitimacy of their existence. If the functions are easily understood and combining them into a single function would introduce complexity or undesirable interactions than it may be better to keep them as separate functions. On the other hand, if a function has been copied by a programmer so they have "ownership" of the code and need not be concerned with potential modifications to the original function would not be acceptable. Another concern is a block of code that is simply copied and repeated x number of times in a function (not referring to loops). Judgement must be exercised as to whether a function should be constructed to provide the purpose. The use of macros is not a concern, even though once compiled the macro is expanded where it is called, since the "source" of the macro lives in one location and if modified the results will be the same in all instances, which cannot be easily assured when multiple lines of code are copied and placed throughout a module.

Dependencies for ADV_INT_EXP.4

ADV_IMP_EXP.1 Implementation Representation of the TSF

ADV_LLD_EXP.3 Complete Low-Level Design

E.9 Rationale for ADV_LLD_EXP

Low-level Design (ADV_LLD)

Editor Note: ***As with ADV_INT, this currently is written focusing on software. The hardware issue is problematic and must be addressed, although it is not obvious that it can be done in the context of this class. The types of activities and characteristics one wishes to see and to describe in software vs. hardware are arguably different, and trying to shoehorn both into a single class will either lead to something that is unmanageable, or slight one of the two areas. We suggest that this issue be seriously discussed at some point, coming up with interim recommendations on how current evaluations are to proceed, as well as a plan for address such issues in the longer term.***

Objectives

285 The low-level design of a TOE provides a description of the internal workings of the TSF in terms of modules, global data, and their interrelationships. The low-level design is a description of how the TSF is implemented to perform its functions, rather than what the TSF provides as is specified in the FSP. The low-level design is closely tied to the actual implementation of the TSF, unlike the high-level design, which could be implementation-independent. The primary goal of the low-level design is an aid in understanding the implementation of the TSF, both by reviewing the text of the low-level design as well as a guide when examining the implementation representation (source code).

Component levelling

286 The components in this family are levelled on the basis of the entity that determines whether a module is security enforcing or security supporting; the degree of detail provided for the modules of the TSF; and degree of formalism required in the presentation.

Application note

287 A module is generally a relatively small architectural unit that can be characterized in terms of the properties discussed in ADV_INT. When both ADV_LLD requirements and ADV_INT requirements are present in a PP or ST, a “module” in terms in of the ADV_LLD requirements refers to the same entity as a “module” for the ADV_INT requirements.

288 A security-enforcing module is a module that directly implements a security functional requirement (SFR) in the ST (with the exception of FPT_SEP and FPT_RVM; see discussion below for these requirements). Such modules will typically implement a security-enforcing TSFI, but some functionality expressed in an SFR (for example, audit and object re-use functionality) may not be directly tied to a single TSFI. Modules that are not security enforcing must support the FPT_SEP and FPT_RVM requirements, and are termed security supporting.

289 It is important to note that the determination of what “directly implements” means is somewhat subjective. In the most narrow sense of the term, it could be interpreted to mean the one or two lines of code that actually perform a comparison, zeroing operation, etc. that implements a requirement. A broader interpretation might be that it includes the module that is invoked in response to a security-enforcing TSFI, and all modules that may be invoked in turn by that module (and so on until the completion of the call). Neither of these interpretations is particularly satisfying, since the narrowness of the first interpretation may lead to important modules being incorrectly categorized as security supporting, while the second leads to modules that are actually not security enforcing being classified as such.

290 Because of this subjectivity, at ADV_LLD.1 the developer is responsible for determining which modules are security supporting and which are security enforcing, supplying detailed information on modules designated security enforcing while supplying less detailed information on security-supporting modules. At ADV_LLD.2, the developer provides detailed documentation on all modules in security enforcing subsystems, thereby providing the evaluation entity the means to

determine which modules are security enforcing (and thus need closer scrutiny) and which are security supporting. At ADV_LLD.3 and ADV_LLD.4, all modules are described in detail, allowing the evaluator to make a comprehensive determination with detailed information as to whether a module is security enforcing or security supporting.

291 A detailed description of a module in the low-level design should be such that one could create an implementation of the module from the low-level design, and the resulting implementation would be 1) identical to the actual TSF implementation in terms of the interfaces presented and used by the module, and 2) algorithmically identical to the TSF module. For instance, the low-level design may describe a block of processing that is looped over a number of times. The actual implementation may be a for loop or a do loop, both of which could be used to implement the algorithm. Conversely, if a module's actual implementation performed a bubble sort, it would be inadequate for the low-level design to specify that the module "performed a sort"; it would have to describe the type of sort that was being performed.

292 Modules not described in detail should be identified and enough information should be supplied so that 1) the evaluation team can determine whether such modules are security supporting or security enforcing, and 2) the evaluation team has the information necessary to complete the analysis required by ADV_INT requirements if components from that family are included in the PP or ST.

293 In the low-level design, modules are described in detail in terms of the interfaces they present; the interfaces they use; global data they access; the function they provide (the purpose); and an algorithmic description of how they provide that function.

294 The interfaces presented by a module are those interfaces used by other modules to invoke the functionality provided. Interfaces are described in terms of how they are invoked, and any values that are returned. This description would include a list of parameters, and descriptions of these parameters. If a parameter were expected to take on a set of values (e.g., a "flag" parameter), the complete set of values the parameter could take on that would have an effect on module processing would be specified. Likewise, parameters representing data structures are described such that each field of the data structure is identified and described. Note that different programming languages may have additional "interfaces" that would be non-obvious; an example would be operator/function overloading in C++. This "implicit interface" in the class description would also be described as part of the low-level design. Note that although a module could present only one interface, it is more common that a module presents a small set of related interfaces.

295 By contrast, interfaces used by a module must be identified such that it can be determined which module is being invoked by the module being described. It must also be clear from the low-level design the algorithmic reason the invoking module is being called. For instance, if Module A is being described, and it uses Module B's bubble sort routine, an inadequate algorithmic description would be "Module A invokes the double_bubble() interface in Module B to perform a bubble sort." An adequate algorithmic description would be "Module A invokes the double_bubble routine with the list of access control entries; double_bubble() will return the entries sorted first on the username, then on the access_allowed field according the following rules..." The detailed description of a module in the low-level design must provide enough detail so that it is clear what effects Module A is expecting from the bubble sort interface. Note that

one method of presenting these called interfaces is via a call tree, and then the algorithmic description can be included in the algorithmic description of the called module.

296 If the implementation makes use of global data, the low-level design must describe the global data, and in the algorithmic descriptions of the modules indicate how the specific global data are used by the module. Global data are identified and described much like parameters of an interface.

297 The purpose of a module should be described indicating what function the module is providing. It should be sufficient so that the reader could get a general idea of what the module's function is in the architecture.

298 As discussed previously, the algorithmic description of the module should describe in an algorithmic fashion the implementation of the module. This can be done in pseudo-code, through flow charts, or (at ADV_LLD.1, .2, or .3) informal text. It discusses how the parameters to the interface, global data, and called functions are used to accomplish the module's function. It notes changes to global data, system state, and return values produced by the module. It is at the level of detail that an implementation could be derived that would be very similar to the actual implementation of the system.

299 It should be noted that source code does not meet the low-level design requirements. Although the low-level design describes the implementation, it is not the implementation. Further, the comments surrounding the source code are not sufficient low-level design if delivered interspersed in the source code. The low-level design must stand on its own, and not depend on source code to provide details that must be provided in the low level design (whether intentionally or unintentionally). However, if the comments were extracted by some automated or manual process to produce the low-level design (independent of the source code statements), they could be found to be acceptable if they met all of the appropriate LLD requirements.

300 The ADV_LLD.*.2E elements within this family define a requirement that the evaluator determine that the low-level design is an accurate and complete instantiation of the TOE security functional requirements. This provides a direct correspondence between the TOE security functional requirements and the low-level design, in addition to the pairwise correspondences required by the ADV_RCR family. Although the evaluator may use the evidence provided in ADV_RCR as an input to making this determination, ADV_RCR cannot be the basis for a positive finding in this area. The requirement for completeness is intended to be relative to the level of abstraction of the low-level design. Note that for this element, FPT_SEP and FPT_RVM are not explicitly analyzed; the analysis for those requirements is done as part of the activity for the ADV_ARC family of components.

301 In the context of the level of formality of the low-level design, informal and semiformal are considered to be hierarchical in nature. Thus, ADV_LLD.2.1C may also be met with either a semiformal or formal low-level design, and ADV_LLD.2.1C may also be met with a formal low-level design.

Dependencies for ADV_ LLD_EXP.4

ADV_HLD_EXP.3 Complete High-Level Design

ADV_IMP_EXP.1 Implementation Representation of the TSF

E.10 Rationale for ADV_RCR_EXP

Representation Correspondence (ADV_RCR)

Objectives

302 The correspondence between the various TSF representations (i.e. security functional requirements, TSFIs specified in the functional specification, high-level design, low-level design, implementation representation) addresses the correct and complete instantiation of the requirements to the least abstract TSF representation provided. The developer provides a correspondence or mapping that provides the evaluators with a roadmap to use in their evaluation activities. The evaluators use the evidence provided in this family along with the evidence required in other families to determine if an accurate and complete step-wise refinement has been achieved.

Component levelling

303 The components in this family are levelled on the basis of the levels of abstraction of the TSF that is required, with the highest component in this family requiring a formal correspondence proof that the SFRs are completely and accurately implemented in the TSFI.

Application note

304 The developer must provide a mapping or correspondence that indicates the how the SFRs are realised in the TSFI. Some of the security relevant functionality specified in the SFRs may not be present in the functional specification's presentation of the TSFIs (e.g., residual information protection may not be discussed in the functional specification for a given interface even though using that interface may involve the allocation of a resource that has been zeroized, domain separation, reference validation). The developer is still expected to map these types of SFRs to the appropriate TSFIs.

305 The developer is expected to map all the TSFI to a corresponding subsystem that is described in the high level design. All of the modules presented in the low-level design are required to be mapped to its corresponding subsystem as well as the implementation representation that realises the modules.

306 A formal proof of correspondence is required in ADV_RCR.3 to demonstrate the SFRs are completely and accurately realised in the TSFIs. This may require the SFRs to be restated in a language that can be used by a tool to formally specify the TSFI or to demonstrate the TSFI (which also may have to be transformed into a formal methods language) fully and accurately implement the SFRs. In these cases, any tools used to demonstrate correspondence or to generate formally specified SFRs or TSFI need to be examined to ensure the correctness of their

application.

Dependencies for ADV_ RCR_EXP.3

ADV_FSP_EXP.6 Formal Functional Specification with Indirect Error Mapping

ADV_HLD_EXP.1 Descriptive High-Level Design

ADV_LLD_EXP.1 Descriptive High-Level Design

ADV_IMP_EXP.1 Implementation Representation of the TSF

E.10 Rationale for ADV_SPM_EXP

Security Policy Modeling (ADV_SPM)

Editor Note: ***The updates in this section are derived only from the Final Interpretations. In discussions with the people who are frighteningly familiar with policy modeling, it became apparent that their use of "semiformal" refers to expressing things formally and then proving their soundness, completeness, etc manually.***

Objectives

307 It is the objective of this family to provide additional assurance that the security functions in the functional specification enforce the policies in the TSP. This is accomplished via the development of a security policy model of the policies of the TSP, and establishing a correspondence between the functional specification and the security policy model of these policies of the TSP.

Component levelling

308 The components in this family are levelled on the basis of the degree of formality required of the TSP model, and the degree of formality required of the correspondence between the TSP model and the functional specification.

Application note

309 A *model* is merely an abstraction or simplification of reality, the purpose of which is to capture key aspects of the behaviour of that reality. An IT s*ecurity policy* is set of restrictions and properties that specify how information and computing resources are prevented from being used

to violate an organizational security policy, accompanied by a persuasive set of engineering arguments showing that these restrictions and properties play a key role in the enforcement of the organizational security policy. A *security policy model* is a precise, possibly formal, presentation of the security policy enforced by the IT system; it identifies the set of rules and practices that regulates how a system manages, protects, and otherwise controls the system resources.

310 The term *security policy* has traditionally been associated with only access or information flow control policies. The CC requirements bear this out: only the requirements for access and information flow control contain assignments to explicitly describe the rules and coverage of these policies. However, a TSP is not limited to access control or information control policies; there are also audit policies, identification policies, authentication policies, encryption policies, management policies, and any other security policies that are enforced by the TOE, as described in the PP/ST. The rules of these policies are not explicitly stated in any requirement; they are implicit in the sense that one must consider all of the related requirement components (e.g. those of the FAU family to derive the implicit audit policy).

311 An informal model is merely a description of these policies. Therefore, any security policy can be stated informally. The only difference between an informal model and a semiformal one is format or presentation (rather than content); therefore, any informal model can be written in a semiformal style.

312 Some policies can be modelled formally, while others cannot. ADV_SPM.3.2D and ADV_SPM.3.3D contain assignments for identifying the policies that are semiformally and formally modelled. Formal TSP models have traditionally represented only subsets of those policies (i.e. the access control and information control policies), because modeling certain policies is currently beyond the state of the art. The current state of the art determines the policies that can be formally modeled, and the PP/ST author should identify specific functions and associated policies that can be, and thus are required to be, formally modeled.

313 The level of formality of the TSP model and of the functional specification determine the necessary level of formality of the correspondence between them: the correspondence is at the same level of formality as the model and the functional specification if they are of the same level of formality, or at the level of the less-formal one if they are different.

314 Because informal, semiformal and formal are considered to be hierarchical in nature, ADV_SPM.1 may also be met with either a semiformal or formal TSP model, and ADV_SPM.2.1C may also be met with a formal TSP model. Furthermore, ADV_SPM.2.4C and ADV_SPM.3.4C may be met with a formal proof of correspondence. Finally, in the absence of any requirements on its level of formality, a demonstration of correspondence may be informal, semiformal or formal.

315 For any security policy where formal models are not possible, the policy must be provided in semiformal form. If none of the TOE's security policies can be formally modelled, ADV_SPM.3 cannot be met.

316 For each of the components within this family, there is a requirement to describe the rules and characteristics of applicable policies of the TSP in the TSP model and to ensure that the TSP model satisfies the corresponding policies of the TSP. The "rules" and "characteristics" of a TSP model are intended to allow flexibility in the type of model that may be developed (e.g. state

transition, non-interference). The security characteristics are the properties that must hold for security to be maintained (e.g., definitions of secure values for TSF data); the description of security rules are how the TOE enforces the values that the security-relevant attributes can take.

317 For example, an authentication policy that uses passwords would include a definition of the minimum acceptable password length in its description within the model. The security characteristics would include the minimum password length; the security rules would describe the TOE's enforcement of that length (should the user attempt to set a password of lesser length, for example).

318 Alternatively, rules may be represented as "properties" (e.g. simple security property) and characteristics may be represented as definitions such as "initial state", "secure state", "subjects" and "objects".

319 The dependency of some functional requirements upon ADV_SPM implies a need for certain contents to be in the model:

- FMT_MTD.3 and FCS_CKM.* require that the model defines secure values for the attributes used in the enforcement of policies that are identified in FMT_MSA.1, which is a dependency. The model would make clear how these values (or combinations thereof) define the security of the TOE.

- FMT_MTD.3 requires that the model defines secure values for the TSF data that are identified in FMT_MTD.1, which is a dependency. The model would make clear how these values (or combinations thereof) define the security of the TOE.

- FPT_FLS.1, FPT_RCV.*, and (indirectly) FRU_FLT.* all require that a secure state be identified, defined, characterized or described in the model. In cases where the model is not state-based, the secure state of the TOE can be defined in terms of the properties that must hold and the values that must be set. This covers every security policy.

Editor Note: ***The CCIMB needs to re-examine the wording of the functional requirements identified in the preceding paragraph. Allegedly the model need not be state-based, but the functional requirements currently read as if the model is state-based. The functional requirements might have to be reworded to be less state-centric; FMT_MSA.1 needs to be rewritten to allow assignment of all policies, rather than only access control and information flow control.***

320 The purpose of the security policy model rationale is to verify that the security functional requirements are consistent with the security policies. It should explain how the effect of each requirement upon each policy was measured and provide a justification for any requirement that is not mapped to any policy.

Dependencies for ADV_SPM_EXP.3

ADV_FSP_EXP.1 Descriptive Functional Specification

Appendix F – Example TOE Scenario

321 For the configuration data and TOE implementation components, Figure F-1 provides a notional illustration of an acceptable scenario for their generation, movement and use, as well as the allocation of components to the TSF and TOE.

322 For simplicity and generality, the entity that develops or modifies a TOE component is called a "TOE developer," even if some of the development is performed by an entity that is an integrator or customer in some other scenario. If a component (e.g., a new hardware dependent module) is to be integrated into the TOE, then the component, as well as the combination of that component with the rest of the TOE will need to have been evaluated. Also, implementation components will always need to be accepted by the TOE trusted delivery mechanism. For example, if an "integrator," receives a TOE from the original developer and then modifies certain TOE components as part of the integration of the SK into a larger component/system – conceptually creating a new TOE -- evaluation or re-evaluation of the new TOE must occur. It is beyond the scope of this protection profile to specify requirements that are specific to a partial re-evaluation of the new TOE.

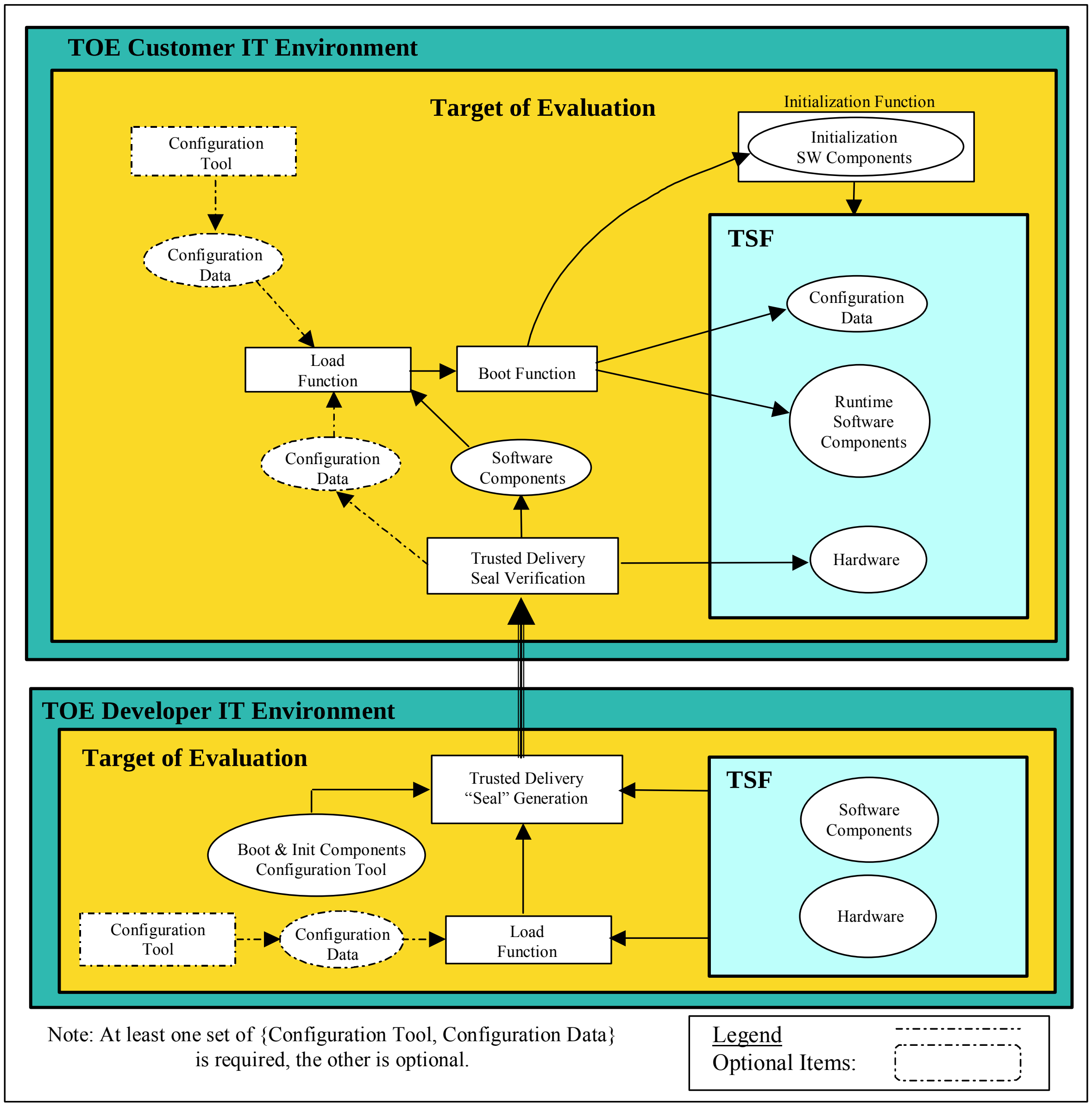

TOE Customer IT Environment
Target of Evaluation
Initialization Function
Initialization SW Components
Configuration Tool
Configuration Data
TSF
Configuration Data
Load Function
Boot Function
Runtime Software Components
Configuration Data
Software Components
Trusted Delivery Seal Verification
Hardware
TOE Developer IT Environment
Target of Evaluation
Trusted Delivery "Seal" Generation
TSF
Software Components
Boot & Init Components Configuration Tool
Hardware
Configuration Tool
Configuration Data
Load Function
Note: At least one set of {Configuration Tool, Configuration Data} is required, the other is optional.
Legend
Optional Items:

Figure F-1 Example TOE Scenario